Euphoria: A Scalable, Event-driven Architecture for Designing Interactions across Heterogeneous Devices in Smart Environments

Article *in* Information and Software Technology · January 2019

DOI: 10.1016/j.infsof.2019.01.006

CITATION
1

READS
79

3 authors:

Ovidiu Andrei Schipor
Stefan cel Mare University of Suceava
32 PUBLICATIONS **159** CITATIONS
SEE PROFILE

Radu-Daniel Vatavu
Stefan cel Mare University of Suceava
98 PUBLICATIONS **1,085** CITATIONS
SEE PROFILE

Jean Vanderdonckt
Université Catholique de Louvain - UCLouvain
483 PUBLICATIONS **7,400** CITATIONS
SEE PROFILE

Information and Software Technology 109 (2019) 43–59

Contents lists available at ScienceDirect

Information and Software Technology

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

Euphoria: A Scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments

Ovidiu-Andrei Schipor[a], Radu-Daniel Vatavu[a,*], Jean Vanderdonckt[b]

[a] MintViz Lab | MANSiD Research Center, University Stefan cel Mare of Suceava 13 Universitatii, Suceava 720229, Romania
[b] Louvain School of Management, Université catholique de Louvain, Louvain-La-Neuve B-1348, Belgium

ARTICLE INFO

Keywords:
Context-aware computing
Mobile computing
Wearable computing
Multi-device interaction
Smart environments
Smart spaces
Software architecture

ABSTRACT

Context: From personal mobile and wearable devices to public ambient displays, our digital ecosystem has been growing with a large variety of smart sensors and devices that can capture and deliver insightful data to connected applications, creating thus the need for new software architectures to enable fluent and flexible interactions in such smart environments.
Objective: We introduce EUPHORIA, a new software architecture design and implementation that enables easy prototyping, deployment, and evaluation of adaptable and flexible interactions across heterogeneous devices in smart environments.
Method: We designed EUPHORIA by following the requirements of the ISO/IEC 25010:2011 standard on Software Quality Requirements and Evaluation applied to the specific context of smart environments.
Results: To demonstrate the adaptability and flexibility of EUPHORIA, we describe three application scenarios for contexts of use involving multiple users, multiple input/output devices, and various types of smart environments, as follows: (1) wearable user interfaces and whole-body gesture input for interacting with public ambient displays, (2) multi-device interactions in physical-digital spaces, and (3) interactions on smartwatches for a connected car application scenario. We also perform a technical evaluation of EUPHORIA regarding the main factors responsible for the magnitudes of the request-response times for producing, broadcasting, and consuming messages inside the architecture. We deliver the source code of EUPHORIA free to download and use for research purposes.
Conclusion: By introducing EUPHORIA and discussing its applicability, we hope to foster advances and developments in new software architecture initiatives for our increasingly complex smart environments, but also to readily support implementations of novel interactive systems and applications for smart environments of all kinds.

1. Introduction

The mobile and wearable computing era presents users with an ever-growing variety of smart devices (*e.g.*, smart rings, smartwatches, intelligent displays, etc.) that, when interconnected to capture and share data in a physical-digital ecology of personal and public devices and applications, acquire the ability to support a large variety of their users' digital needs [1]. Personal mobile and wearable devices as well as public interactive systems (*e.g.*, ambient displays, interactive floors, etc.) embed increasingly sophisticated sensors and algorithms that have practically remodeled our digital ecosystem. Moreover, software services distributed over the web have led to the emergence of a new form of physical-digital reality, in which asynchronous data collection, processing, and interpretation provided by heterogeneous devices (*e.g.*, devices with different operating systems, communication protocols, Internet-of-Things standards, etc.) have become the foundation of smart environments [2]. Consequently, end-users are confronted with a diverse palette of contexts of use [3] defined by the many and complex relationships that can be established between users, personal devices, and public interactive systems. For example, Fig. 1 illustrates a context of use for a smart environment with a variety of devices being employed by three users to transfer and display content from their own personal devices onto a wall display via simple touches and pointing gestures. Integrating this variety of devices, each with their own operating system, communication protocols, software development kits and supported programming languages, technical characteristics (*e.g.*, data frames per second, latency), and available interaction techniques (*e.g.*, touch, voice, gesture, etc.), is a considerable design and implementation effort that should not transpire to end-users, but rather hide behind flexible and intuitive interactions, *e.g.*, touching a digital picture on the smartphone followed by

* Corresponding author.
E-mail addresses: schipor@eed.usv.ro (O.-A. Schipor), radu.vatavu@usm.ro (R.-D. Vatavu), jean.vanderdonckt@uclouvain.be (J. Vanderdonckt).
URL: http://www.eed.usv.ro/~vatavu (R.-D. Vatavu)

https://doi.org/10.1016/j.infsof.2019.01.006
Received 25 August 2018; Received in revised form 15 January 2019; Accepted 18 January 2019
Available online 28 January 2019

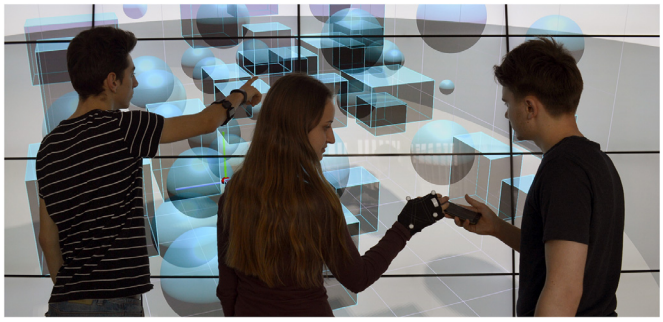


Fig. 1. An illustrative example of a context of use for a smart environment depicting a multi-user, multi-device scenario: content is shared between users' personal devices and the wall display via simple touches and pointing actions. EUPHORIA, our new software architecture, enables simple integration of all the devices illustrated in this image (*i.e.*, smartphone, smartwatch, armband, wall display), and many other devices as well, via a scalable, event-driven approach based on common web protocols and technology.

pointing the same hand towards the wall display should readily perform the transfer [4–6].

Although smart environments, such as the example depicted in Fig. 1, are still in their infancy, their immense potential to enable intuitive, fluent, and flexible interactions between users, devices, applications, and systems is expected to ultimately integrate user behavior [7,8]. Consequently, there has been a long-standing interest in the community towards designing hardware and software infrastructure to underpin the functionality requirements of such environments; see Fries [9], Schipor *et al.* [10], Vatavu *et al.* [11], or Lou *et al.* [12] for a few examples. Among the preferable design criteria for software architectures for smart environments are modularity, scalability, and asynchronicity for producing, processing, and transmitting messages and events, which have become fundamental to effective design and efficient implementation of interactions in smart environments. Nevertheless, perpetual advances towards more relevant standards, such as for the Internet-of-Things and connected devices, accompanied by the unprecedented diversity of smart wearables [13–18], has fostered the need for new hardware and software architecture designs. Consequently, there is still room for innovation in terms of effective and efficient software architectures to support interactions in smart environments.

In this paper, we address new software architecture designs for smart environments of various kinds (*e.g.*, smart rooms, in-vehicle interactive systems, on-body networks, etc.) to enable adaptable and flexible interactions and to accommodate a wide diversity of current, but also future input/output devices. Specifically, we target software architecture designs providing asynchronous, event-driven communication between heterogeneous devices in ways that are simple for the designer to prototype, the developer to implement, and the end-user to interact with. Toward this end, we developed EUPHORIA,[1] a new scalable, event-driven software architecture enabling heterogeneous and asynchronous communications between personal devices and public interactive installations and services that constitute a smart environment. To make EUPHORIA as flexible and interoperable as possible [17,19–24], we made use of web protocols and web technology, such as `http` and the interchangeable `JSON` data format, that we leverage to produce, transmit, process, and interpret device- and environment-specific events through light software interfaces. For example, exposing a custom event in EUPHORIA is equivalent to running a standard `http` request, while consuming events is readily achieved by subscribing to event notifications, implementable in a few lines of `JavaScript` code. The main characteristics of EUPHORIA are its *flexibility, asynchronicity*, and support for *device heterogeneousness*, which foster accommodation of a wide range of current and future input and output devices. Our contributions in this paper are as follows:

1. We introduce EUPHORIA, a new software architecture design and implementation with asynchronous event processing for designing flexible interactions with digital content across heterogeneous devices in smart environments. EUPHORIA implements device- and environment-specific event-driven production, sharing, and processing using web protocols and technology readily accessible from any web-connected device, such as smartphones, tablets, smartwatches, etc.
2. To demonstrate EUPHORIA's effectiveness for practical use and adoption in smart environments of various kinds, we describe three application scenarios involving a variety of devices representative for today's smart technology, *e.g.*, smart rings, smartwatches, smartphones, motion tracking sensors, a multitouch tabletop, and a wall display.
3. To foster and encourage further developments in the community toward efficient software architectures for smart environments, we deliver EUPHORIA to researchers and practitioners as an open-source architecture, driving interactions for smart environments with event-based communications and processing. EUPHORIA is available for free for research purposes at the permanent web address **http://www.eed.usv.ro/mintviz/resources/Euphoria/** together with a live demonstration, a sandbox for testing messages, and examples of use.

2. Related work

In this section, we discuss prior work in software architecture design related to supporting interactions in smart environments. We rely on this prior work to inform the design principles, approach, and technical requirements of the EUPHORIA software architecture.

2.1. Event-driven software architecture design

EUPHORIA implements the principles and concepts of event-driven design to support event-response systems [25], while it also shares other characteristics with object-oriented, client-server, and layered software architectures, as we discuss later in this article. To motivate the need for and the usefulness of the event-driven design approach, we start with a brief overview of the related work in this area.

The term "software architecture" generally refers to an abstract, yet implementable, structural and functional description of a software system [26]. Due to the central role played by the software architecture in the lifetime of an application, a vast amount of literature has addressed this topic, highlighting several main architectural styles as well as various combinations and adaptations [27], including meta-models [28]. For example, in the area of engineering interactive systems, the Arch/Slinky meta-model [29] structures the software architecture of any interactive system into five components: (1) a dialog controller that ensures the sequencing of asynchronous events from/to (2) the functional core of the application and from/to (3) the interaction toolkit, respectively, via (4) a functional core adapter and (5) a presentation adapter. Acting from the functional core, the architecture directly determines the general behavior, limits, and advantages of the final software product [29]. From this perspective, a software architecture usually consists of several components that are connected through data flows [30].

A first attempt to introduce event-driven software architectures (EDA) belongs to Yourdon and Constantine [31], who were the first to coin the concept of the transaction center as a software component that is *"able to get (obtain or respond to) transactions in raw form, analyze each transaction to determine its type, dispatch on type of transaction,* [and] *complete the processing of each transaction"* (p. 225). EDA architectures consist of four layers: producers, emitters, an engine, and consumers;

[1] The name EUPHORIA stands for Event-based Unified Platform for HeteRogeneous and Asynchronous Interactions.

see Moxey *et al.* [32]. A formal distinction is made between the concept of an "event," a change in state that occurs at a precise moment in time at the producer level, and that of a "message," an event notification that travels from the emitter to the consumer [33]. By construction, an EDA is well suited for implementing applications supporting low coupling between components [34] that communicate with each other asynchronously [35,36].

The EDA approach has been demonstrated suitable for a wide range of applications and especially for highly interactive systems [37], from networked sensors [38,39] and smart grids [40,41] to handling business process management systems [42,43] and running operations in smart cities [44,45]. Three-dimensional visual reconstruction using trains of events with very high temporal resolution [46], simulations of spiking neural networks [47], and integration of multi-agent systems [48] are among other recent applications for which EDA architectures played an important role. All these applications rely on the same set of design requirements, including responsiveness, asynchronicity of events, and heterogeneousness of data sources.

2.2. Software architectures for smart environments

Previous work provided various definitions of what a smart environment is from the perspectives of creating visions for ubiquitous computing [1,49] and ambient intelligence [50], but also for providing practical guidelines for designing the user experience in such environments [51]. In this work, we rely on the definition of Cook and Das [52], according to which a smart environment is *"a small world where all kinds of smart devices are continuously working to make inhabitants' lives more comfortable"* (p. 3) Particularly, we address the aspect of implementing smart environments by focusing on a specific part of this definition that refers to *"all kinds of smart devices,"* their interoperability and ability to communicate seamlessly, but also we restrict our interest and application domain to the *"small world"*, as expressed by Cook and Das [52], where usually tens and up to one hundred of such devices are active. From this perspective, we don't address architectures designed for large geographical areas, such as smart cities. Furthermore, we connect to the variety and richness of devices from the vision described by Weiser *et al.* [49] of a *"physical world richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives and connected through a continuous network"* (p. 694). To this end, EUPHORIA addresses the practical aspects of implementing applications for smart environments by integrating the functional and operational diversity of heterogeneous devices into flexible, adaptable interactions with support for asynchronicity.

Special attention has been devoted to EDAs for the conceptual foundation of smart environments [22,44,53]. EDAs are suitable for such applications since smart environments consist of numerous heterogeneous sensors and devices interacting with each other in ways that are not always deterministic. From wearable sensors and gadgets, such as smart rings [14], finger augmentation devices [54], and armbands [55] to smartwatches [18,19], health-monitoring smart belts [56], and smart clothing [57], there is a wide range of mobile, wearable, and embedded technology to integrate smart environments. Moreover, these devices need to communicate with complex installations, such as systems that track people's location and movement inside the environment [58], producers of multimedia content [59], or very large systems implementing crowd-computer interaction [60] and mass-computer interaction [61]. Low-level parameters provided by devices and applications within a smart environment can be further used to access high-level, contextual information on behalf of context-aware software [62,63]. For example, reliable information about the health or the emotional state of a user usually emerges from more than one sensor. To this end, multiple variables need to be measured across various channels, such as physiological data (*e.g.*, blood pressure, skin conductance, respiratory rhythm), audio (*e.g.*, the tone or the pitch of the user's voice), and video (*e.g.*, the face expression, body posture, or the body gestures of the user); see, for example, Schipor [64] or Gheran *et al.* [14]. Collecting, processing, integrating, and interpreting such complex data in real-time in a unified manner becomes critical to realize the transition from raw data to information and knowledge. As EDA can handle heterogeneous and asynchronous data collection and processing, they represent an appropriate choice for implementing the software components of a smart environment.

There has been a significant trend in the community to provide high-level information, such as emotion [64,65], activity [66,67], or gestures [12,68] through abstract software components in smart environments. For instance, the Gesture Profile for Web Services (GPWS) of Vatavu *et al.* [11] is an event-driven architecture that supports development of gesture interfaces for smart environments by integrating third-party gesture recognition services. Such an approach, specific to software-oriented architecture design, hides from the developer all the complex pattern recognition techniques required to detect and recognize gesture input and allows developers to focus on prototyping interactions. More recently, Lou *et al.* [12] developed Gesture Services for Cyber-Physical Environments (GS-CPE) by extending the GPWS architecture [11]. GS-CPE is a novel, event-driven, service-oriented framework that implements personalized gesture recognition and user identification. Schipor *et al.* [10] introduced a software architecture to implement interactions with spatially-indexed media in smart environments that instantiate mappings between digital content and specific regions of the physical space. The event-driven architecture design approach has also been applied for real-time activity recognition within a smart home [66] and for collaborative environments [24]. The flexibility of event-driven architectures enables readily integration of data processing and interpretation algorithms into the functional core of applications, such as specialized detectors to assess the similarity of activity events or complex machine learning approaches, such as Support-Vector Machines (SVM), for recognizing users' activity patterns in real-time [66].

2.3. Designing interactions for smart environments

Interacting in smart environments poses many challenges. While people can interact with each other through speech and body language, devices need more formal contexts and structures for communication. For this reason, prior work has explored natural language interpretation [69], gestures and whole-body movements [68], emotion recognition [65,70], and the interpretation of electrical brain signals [71] to address these challenges.

The "nomadic gestures" concept and implementation of Vatavu [72] introduced a shift of perspective in terms of the reusability of gesture commands across ambient interactive systems, which can have access to their users' gesture preferences on the fly and adapt their internal gesture set accordingly. "Smart-Pockets" [6] is another attempt to make interactions between users and ambient interactive systems simple and intuitive. With Smart-Pockets and their extension to "Smart-Containers," users can access their personal digital content by pointing to various pockets on their clothes or accessories, such as bags [6]. The short access time (similar in magnitude to the production time of touch gestures) and the high accuracy rates for recognizing Smart-Pockets actions make this technique viable for retrieving digital content effectively for visualization on public ambient displays. Later in this paper, we show how EUPHORIA can be readily used to implement Smart-Pockets interactions.

The continuous miniaturization of computing devices has created the need to investigate new interaction modalities. For example, Funk *et al.* [73] employed a touch-sensitive wristband to enable text entry on a smartwatch. The authors showed that a multi-tap keyboard layout superimposed on the wristband was appropriate for entering short text. Hands-free interaction with small wearables was investigated as well [74]. Gaze interaction for smartwatches by tracking eye movements [20] has also proved to be a robust and highly accurate technique. In addition, smartwatches have been used to assist interactions with

smartphones. For instance, the DUET system [19] coordinates motion and touch input on a smartwatch and a smartphone in order to extend their visual and tactile output. However, despite innovative solutions and promising results, such previous attempts have focused on immediate functionality, leaving room for many improvements in terms of the interoperability, standardization, and scalability of the interaction.

Some smart environments need to operate a large number of sensors and devices and, thus, networks of wireless sensors have been proposed [75]. For instance, the Mobile Agent Platform for Sun SPOT of Aiello *et al.* [76] implemented mobile agents to enable flexible programming of interaction rules within an ecosystem of sensors. The Vi-SMARt system [24] showed that the combination of smart environments and multi-agent systems was an appropriate solution for integrating various combinations of sensors. The open-source platform of Bellifemine *et al.* [77] and the BodyCloud software system of Fortino *et al.* [17] focused on sensor networks located on the user's body. BMF-WSANs, a Building Management Framework for Wireless Sensor and Actuator Networks [78], is another platform that allows rapid development and flexible management of monitoring applications in smart environments. Another step forward was implemented with a ready to use platform as a starting point for complex smart infrastructure: the Future Internet Lab (FIWARE Lab) represents a management solution for cloud federation [79]. Several federation management tools, such as Pegasus (Platform as a Service Component), DCA (Deployment and Configuration Adapter), and Monitoring Middleware work together in order to minimize development and deployment effort. However, such solutions target an application domain beyond EUPHORIA's goals, such as systems composed of processing nodes that run on four continents [80].

A promising approach to designing novel interactions in smart environments relies on the ability of smart devices to sense their surroundings, such as to enable "proxemic interaction," a concept first employed to address inter-human communication [81]. For example, a hand-held device can exploit spatial relationships, *e.g.*, orientation and distance, between itself and its users, other devices, and even uninstrumented physical objects in its vicinity in order to establish a communication context [82]. This context further enables a dynamic form of proxemic-aware knowledge and control in relation to the surrounding physical entities and devices [83]. For example, Ledo *et al.* [84] employed the "Proximity Toolkit" [82] and a Vicon motion tracking system to design remote controls for ubiquitous computing ecologies. Their work investigated several interactive scenarios and showed the reliability of such a framework for interacting with appliances and devices from a smart environment.

A smart environment built on top of an EDA architecture can leverage the advantages of proxemic awareness since it is able to detect and integrate state changes in a unified manner [23]. The spatial characteristics of entities that interact in a smart environment can be measured using various techniques. For example, localization of users can be achieved with video cameras, such as the Microsoft Kinect sensor [92]. The XDKinect framework [23] is an example of an architecture that facilitates development of cross-device applications by delivering a time-based application programming interface (API) for whole-body gesture interaction. XDKinect also features a configurable, multi-modal software module to capture gesture and speech commands. The Microsoft Kinect system was also used with wearable sensors in the Mobile@Old platform [85] to assist elderly people to maintain a healthy lifestyle in the comfort of their own homes.

3. Design and technical requirements for Euphoria

We introduce in this section the conceptual framework of our design approach for EUPHORIA, which builds on prior work, but also differentiates from current architectures in a number of ways. To better understand how EUPHORIA positions in the context of the relevant literature and recent developments in terms of software architecture design, we describe in this section a set of specific criteria for EUPHORIA regarding its software quality and contextual properties.

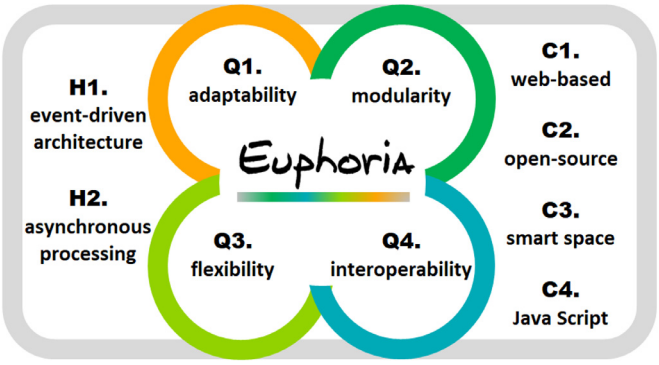


Fig. 2. Visual overview of the two handling techniques (H1 - H2), the four quality properties (Q1 - Q4), and the four contextual properties (C1 - C4) adopted for the design of the EUPHORIA software architecture.

We designed EUPHORIA for handling events and messages produced and consumed by devices and software applications that constitute smart environments according to the definition of Cook and Das [52] that are typical for implementing smart rooms, on-body networks, or interactions for in-vehicle environments and connected cars, to name a few examples. For such environments, the number of devices is usually small, but the design goal of EUPHORIA is to enable interactive responses for end-users by processing messages and events produced and consumed by tens of devices and sensors, up to one hundred different devices operating simultaneously. To this end, we adopted several handling techniques and quality properties for EUPHORIA; see Fig. 2 for a visual illustration.

H1. **The Event-Response System (ERS) technique** for multi-threaded dialogues [25] enables physical, hardware, and software changes that occur in a smart environment to be readily detected and integrated in the business logic of EUPHORIA. By adopting this technique, events can be generated by end-users, software and hardware components and handled through event listeners in a way that preserves the decomposition principles listed by Parnas [34], *i.e.*, high level of abstraction, high internal cohesion, and low coupling.

H2. **Asynchronicity of data processing** [9] enables appropriate handling of unpredictable time-response behavior of the entities within the smart environment.

We also adopted the following four quality properties for EUPHORIA by relating to the Software Quality Requirements and Evaluation (SQuaRE) **ISO/IEC** 25000 series of standards [86,87]:

Q1. **Adaptability**, which subsumes **Scalability**, measures the ability of a system to adjust effectively and efficiently to changes occurring in its context of use, *e.g.*, new hardware or software modules added or removed from the system. Adaptability represents a key characteristic of EUPHORIA which, by design, needs to accommodate effectively various types of input/output devices for various types of smart environments. Two levels of scalability are especially relevant for the smart environments addressed by EUPHORIA: (1) scalability with respect to the size of the messages that are being produced, exchanged, and processed within the smart environment, *e.g.*, messages incorporating data about users moving in the physical space or data about users' interactions performed on specific devices, and (2) scalability with respect to the number of producers and consumers from/to which these messages are created/addressed. We refer to these two types of scalability as *data-oriented* and *environment-oriented* scalability.

Q2. **Modularity** evaluates the degree to which changes occurring in one component of the system determine changes in other com-

ponents, which is important for low coupling [34]. Modularity is a key feature for EUPHORIA to facilitate integration of diverse input/output devices.

Q3. **Flexibility** represents the system ability to be used reliably in new contexts of use, particularly contexts that were not specified in the original requirements, without any external intervention or with a limited intervention only. EUPHORIA should be flexible to implement a variety of application scenarios for smart environments, including smart rooms, on-body networks, in-vehicle environments, etc., with as little changes as possible in its architecture.

Q4. **Interoperability** refers to the degree in which two or more systems are able to work together toward a common goal, such as exchanging or synchronizing data. Interoperability enables a heterogeneous smart environment composed of different devices to operate effectively.

A set of four contextual properties was also adopted in order to shape the EUPHORIA architecture to the specific needs of designing flexible and adaptable interaction in smart environments, as follows:

C1. **Web-based** refers to adherence to common Internet standards and protocols for data processing and exchanging information through messages, such as the `JSON` data format.

C2. **Adoption of open source standards and technology** refers to relying on open source technologies that preserve public accessibility and free usage of the smart environment.

C3. **Smart environment orientation** refers to the capability of supporting interactions between the three key aspects of a smart environment: users, personal devices, and public systems and installations.

C4. **JavaScript orientation** refers to the usage of `JavaScript`, one of the three core technologies of the web alongside `HTML` and `CSS`, as the default description and programming language shared by all the modules of EUPHORIA. However, beyond this capability, any language that implements `WebSockets` can readily be used to connect to and use the features of the EUPHORIA architecture.

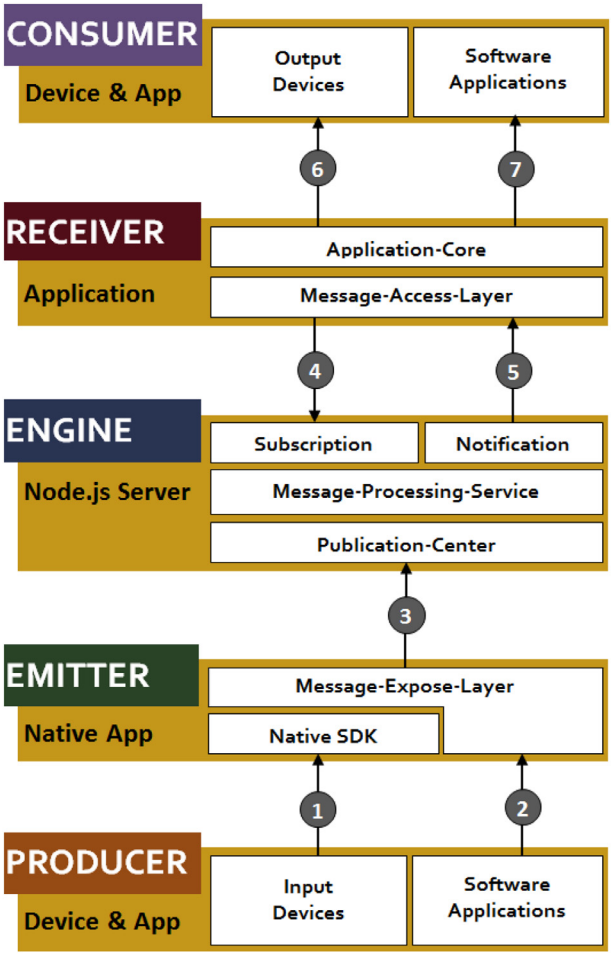


Fig. 3. The multi-layer architecture of EUPHORIA, consisting in event producers, emitters, a processing engine, and event receivers and consumers. Numbers ❶ to ❼ represent dataflows that are discussed in the text.

4. The Euphoria software architecture

We present in the following the blueprint of the EUPHORIA software architecture by focusing on technical details that are relevant to smart environments, such as the environment depicted in Fig. 1. The core design approach of EUPHORIA revolves around the production, transmission, detection, and consumption of device- and environment-specific events, which we define as significant changes in the state of a hardware or software module that is part of the smart environment [33]. Fig. 3 illustrates the architecture adopted for EUPHORIA with its main components: PRODUCERS, EMITTERS, PROCESSING-ENGINE, RECEIVERS, and CONSUMERS, according to the model of event-driven architectures of Moxey *et al.* [32].

We designed EUPHORIA by implementing the separation of concerns between its functional blocks (*i.e.*, the quality property Q2) and the thickness of its adaptation layers (quality property Q1). Also, following the principles of event-driven software architecture design (H1) [31–33], the information about events triggered by PRODUCERS is packed in the form of messages and exposed by EMITTERS to the next layer. Then, messages are processed and dispatched by the PROCESSING-ENGINE to third-party RECEIVERS and CONSUMERS that will interpret the incoming data according to their inner business logic and take appropriate actions in the smart environment, *e.g.*, an event detected by a smartwatch worn by the user on their wrist can result in an item being selected on the large display in front of the user; see the example depicted in Fig. 1. A message transmitted through EUPHORIA consists in a *header* with metadata (*e.g.*, the device name, device address, and the name of the event) and a message *body* that contains the relevant data for the event. Fig. 4 illustrates four examples of messages of various complexities produced by a wearable armband, a smartphone, and two environmental sensors.

The PRODUCER is the component for which a relevant change in state is notified to the architecture in the form of an *event*. Producers are instantiated by input devices (*e.g.*, motion sensors from the environment, smart mobile devices and wearables, etc.) or by software services (*e.g.*, a particular gesture command has been detected for a particular user by a gesture recognition service from the web). For PRODUCERS that are instantiated by input devices, events refer to any physical change in their states that are relevant to the environment, such as the press of a button or a change in the orientation of the wrist for a smartwatch. Software events can also be produced in relation to changes in the condition of some software application. Software events prove useful for debugging phases, running simulations for the environment, or for integrating complex software logic provided by third-party libraries and services, such as those available on the web. The layers of EUPHORIA are unaware of the inner working details of the other layers (*i.e.*, the quality property Q2). For example, it is not important for PRODUCERS to know which components, if any, will process and consume the events they produce. This feature facilitates seamless integration of new devices and software applications into EUPHORIA, including those yet to appear or be developed in the future, such as devices for mobile augmented reality [88], without any modification in EUPHORIA's main core.

Once an event has been produced, the event triggers the instantiation of a message (see dataflows ❶ and ❷ in Fig. 3), which travels to

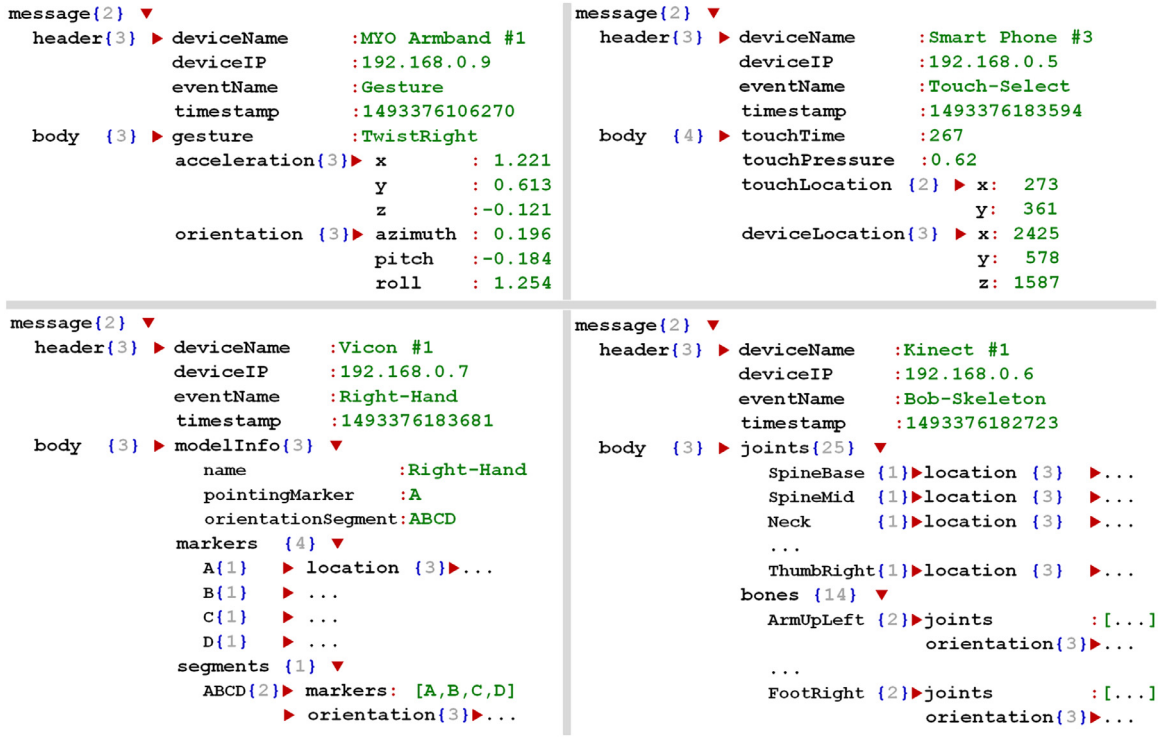


Fig. 4. Examples of JSON messages of various complexity: a message containing acceleration and orientation data collected from a Myo armband (top left); touch input data reported by a smartphone (top right); motion data tracked by a Vicon system (bottom left); and a message containing a Microsoft Kinect skeleton (bottom right).

the next levels of the architecture. In the case the event was produced by an input device, the EMITTER is built around that device's software development kit (SDK) that delivers the required software components to collect the specific parameters of the event (*e.g.*, the properties of a touch on a touchscreen, the orientation of a motion-sensing device, etc.; see the examples illustrated in Fig. 4). The EMITTER collects event data and packs the data in the form of messages which are delivered to the higher layers of the architecture. In most cases, the Message-Expose-Layer (see Fig. 3) needs to be implemented in native code, although some SDKs or software applications may already offer high-level interfacing options for a given input device. The EMITTER and RECEIVER represent the interface between the EUPHORIA core and the variety of devices and software applications that constitute the smart environment. In order to expose or receive a message, a software conversion between the PRODUCER or CONSUMER's centric model is needed to the generic model implemented by EUPHORIA. The utilization of well established technology, such as `http` and JSON, creates an adaptation layer as thin as possible that conserves flexibility and interoperability. In fact, the EMITTER and RECEIVER are the only levels of the architecture that need to be implemented by developers. Since the EMITTER component is required for every input device, we designed it as general and reusable as possible (according to the quality properties Q3 and Q4). In a raw form, the EMITTER only consists in the code necessary to encapsulate data in the JSON representation and to send this message via an `http` request. Feeding our architecture with an event is as simple as requesting content from a Uniform Resource Locator (URL): the query string represents the message.

The PROCESSING-ENGINE is the core component of EUPHORIA. Its main responsibility is to receive messages from EMITTERS (see dataflow ❸ in Fig. 3) and to dispatch those messages to the appropriate CONSUMERS that have registered to receive specific types of events. In our implementation, the PROCESSING-ENGINE is run by a `node.js` web server.[2] The event messages collected by the Publication-Center are unpacked by the Message-Processing-Service; see Fig. 3. Before receiving any notification, a new CONSUMER must register with the PROCESSING-ENGINE by following the protocol of a Subscription module. As part of the subscription protocol, the CONSUMER will specify the type of events relevant to its inner logic and the corresponding PRODUCERS. Once registration is complete, the PROCESSING-ENGINE will automatically notify the CONSUMER about those events that meet the requirements agreed during subscription. By performing both registration and notification, the PROCESSING-ENGINE and the CONSUMER rely on `WebSockets` mechanisms, a set of communication protocols that enable full-duplex, TCP-based sessions between a client and a server [89,90]. The Message-Processing-Service (Fig. 3) routes the messages to the appropriate CONSUMERS and converts the data in the JSON interchangeable format (*i.e.*, the quality property Q4). The PROCESSING-ENGINE is completely unaware about the internal structure of producers and consumers. This abstraction supports both scalability (Q1) and the ability to integrate heterogeneous input devices (Q4).

The RECEIVER is any third-party software application that requests events from the smart environment to run its functional core. The RECEIVER establishes a `WebSocket` connection with the PROCESSING-ENGINE using the Message-Access-Layer (see dataflows ❹ and ❺ in Fig. 3). In addition to the filter list of the types of events and associated PRODUCERS, the RECEIVER can also specify other optional parameters, such as a minimum time interval between receiving two consecutive events of the same type to prevent intractable accumulations of events to handle at once. This practice also serves to prevent the overloading of the network with unnecessary or redundant traffic. One important goal is to maintain a very thin layer (low coupling) between a RECEIVER and the PROCESSING-ENGINE so that any existing software application can integrate with EUPHORIA with a minimum coding effort (Q1, Q2).

[2] https://nodejs.org/en/

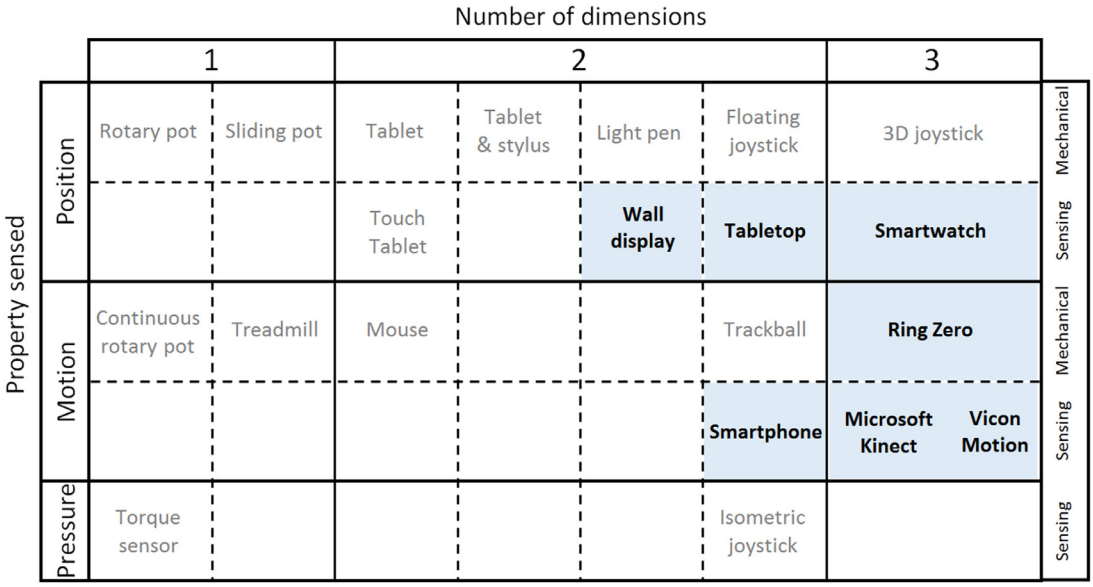


Fig. 5. Devices employed by our application scenarios for EUPHORIA (shown in blue) and their classification according to Buxton's taxonomy [91]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

To this end, we decided to implement the Message-Access-Layer using `WebSockets`, enabling thus real-time and asynchronous data transfer (H1) between the PROCESSING-ENGINE and the RECEIVER. Once received, the message triggers a specific action on the CONSUMER which is the fifth layer of the architecture (see dataflows ❻ and ❼ in Fig. 3). This tier consists in output devices and software applications with which users interact directly. The implementation details of CONSUMERS are locked within the RECEIVER layer to make the ENGINE as decoupled as possible.

5. Application scenarios for Euphoria

We present in this section several application scenarios for EUPHORIA to highlight its capability for prototyping and implementing interactions in smart environments of various kinds. Specifically, we present three application scenarios: (1) wearable user interfaces and whole-body gesture input for interacting with public ambient displays, (2) multi-device interactions in physical-digital spaces, and (3) delivering notifications in smart, connected cars. We hand-picked these scenarios for their coverage of several smart devices, *e.g.*, smartwatch, smart ring, multitouch table, wall display, etc.; see Fig. 5 for a representation of these devices according to Buxton's taxonomy [91] in terms of the number of dimensions supported by each device and the physical properties that are being sensed, *i.e.*, position, motion, or pressure. Next, we briefly describe each application scenario and show how EUPHORIA can be used to support technical implementation in terms of producers, consumers, and message formats for events.

5.1. Smart-Pockets: Fast access to personal data for ambient interactions

Smart-Pockets represent a set of whole-body gestures that enables users to access their personal digital content efficiently for visualization on public ambient displays [6]. A Smart-Pockets gesture is defined as a movement of the hand entailing access to some pocket, such as the trousers left pocket or the right inner breast pocket, followed by a pointing movement of the same hand toward the ambient display. A complementary hand movement from the display to the location of the pocket stores the respective content back into the user's pocket for later retrieval. Vatavu [6] showed that Smart-Pockets interactions are fast, can be recognized robustly even under user-independent training, and are flexible and easily extensible to other containers, such as bags and hand-held objects, denoted as "Smart-Containers" [6]. The pocket metaphor implemented by Smart-Pockets creates a link between physical objects and personal digital content, useful for fast access to data in smart environments. In the following, we show how Smart-Pockets can be readily implemented with EUPHORIA.

Following the description from Vatavu [6, p. 6], users' whole-body movements are captured with the Microsoft Kinect sensor, which reports, among others, the position and movement of the user's body in 3-D as a sequence of skeleton data; see [92] for details. In our implementation, we opted for the "explicit segmentation" approach advocated in Vatavu [6, p. 14] as the preferred detection technique for Smart-Pockets gestures, where users decide when the system should interpret their whole-body movements by signaling explicit "start" and "stop recording" commands. Vatavu [6] discussed that any device that can deliver start/stop events to the system, such as through-fabric capacitive touch input, mid-air devices, or specific hand poses, represents a good candidate to segment Smart-Pockets gestures from continuous whole-body movement (p. 14). For the illustration of this specific use case, Smart-Pockets gestures are segmented with taps performed on the Ring ZERO device [93], an interactive smart ring that reports finger motion and touch events; see Fig. 6, top.

Fig. 6, bottom illustrates the instantiation of EUPHORIA for this specific implementation as well as examples of event messages. There are two PRODUCERS to implement for the Smart-Pockets scenario, corresponding to the two input devices (the Kinect sensor and the smart ring), and one CONSUMER, represented by the ambient display. The smart ring PRODUCER reports to EUPHORIA taps performed on the ring's button to be interpreted as start and stop events for the purpose of gesture segmentation. The Ring ZERO device connects wirelessly to a smart device, where a custom Android application receives ring events via Bluetooth 4.0, filters those events, and sends taps to EUPHORIA. The Kinect sensor PRODUCER collects the user's whole-body gestures and reports them to EUPHORIA. The CONSUMER subscribes to both taps and gesture events and implements the Smart-Pockets recognition logic: the whole-body gesture recorded between two consecutive start and stop taps on the smart ring is matched against a set of Smart-Pockets gesture templates. The recognition algorithm is described by Vatavu [6] in detail, so we won't insist on it here. If the match between the candidate gesture and any of the templates is successful, a corresponding action is executed,

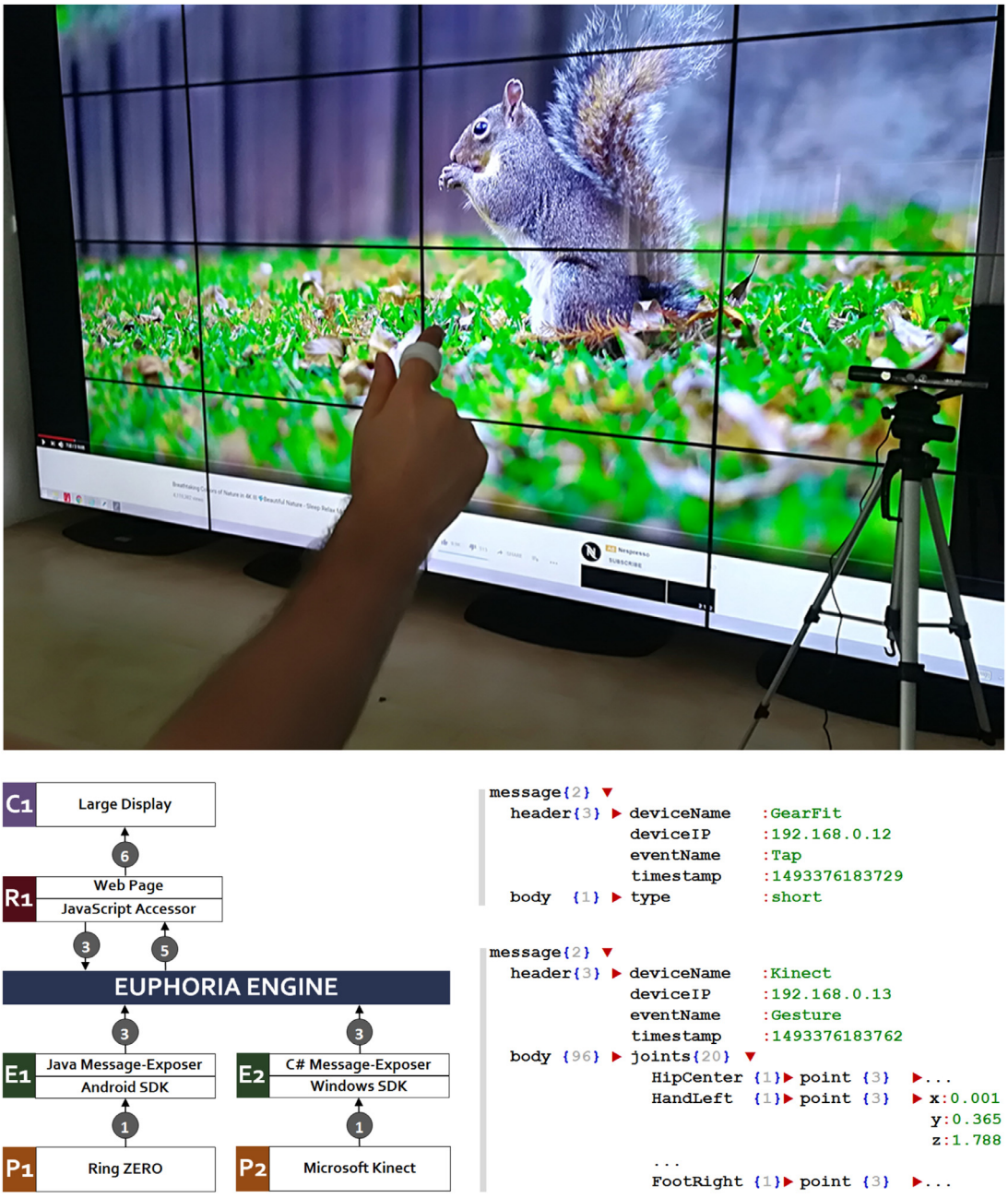


Fig. 6. EUPHORIA applied to implement the Smart-Pockets technique of Vatavu [6]: hand gestures captured by a Microsoft Kinect sensor and segmented using taps on a smart ring (top image) are processed as events through the EUPHORIA architecture (bottom left) implemented using JSON messages (bottom right). In this example, the user performs a Smart-Pockets gesture to visualize on a large ambient display a YouTube video, for which the link was stored in one of their pockets.

such as display the content associated to the specific pocket onto the large ambient display. Messages travel in EUPHORIA from the two input devices to the ENGINE thorough a native software layer implemented in Java for Android in the case of the Ring ZERO device and in C# for Windows for the Microsoft Kinect sensor. An HTML web page implements the CONSUMER represented by the public display that plays YouTube videos of links stored in user's smart pockets.

5.2. Access to multimedia digital content anchored to physical locations in a smart environment

New physical-digital interfaces that enable users to access digital content mapped to the physical space, such as spatial augmented reality [94,95], imaginary interfaces [13,54,96], or physical loci [97], have introduced a new perspective on the interplay between our digital and physical realities. A digital layer superimposed upon the topography of the physical environment unlocks new possibilities to access and operate digital content. For instance, digital objects can be dynamically associated with specific regions of the physical space and accessed with a mere pointing of the hand. Multiple digital layers can also coexist at once in the same physical space leading to different experiences for different users.

To demonstrate how EUPHORIA can be used to readily implement such interactive scenarios, but also how multiple devices can render different views of the same content, we designed an instantiation of EUPHORIA with one PRODUCER and three CONSUMERS, as follows. A 6-camera Vicon motion capture system [98] was employed to track users and their pointing actions inside a physical area of 4 m × 4 m. Three

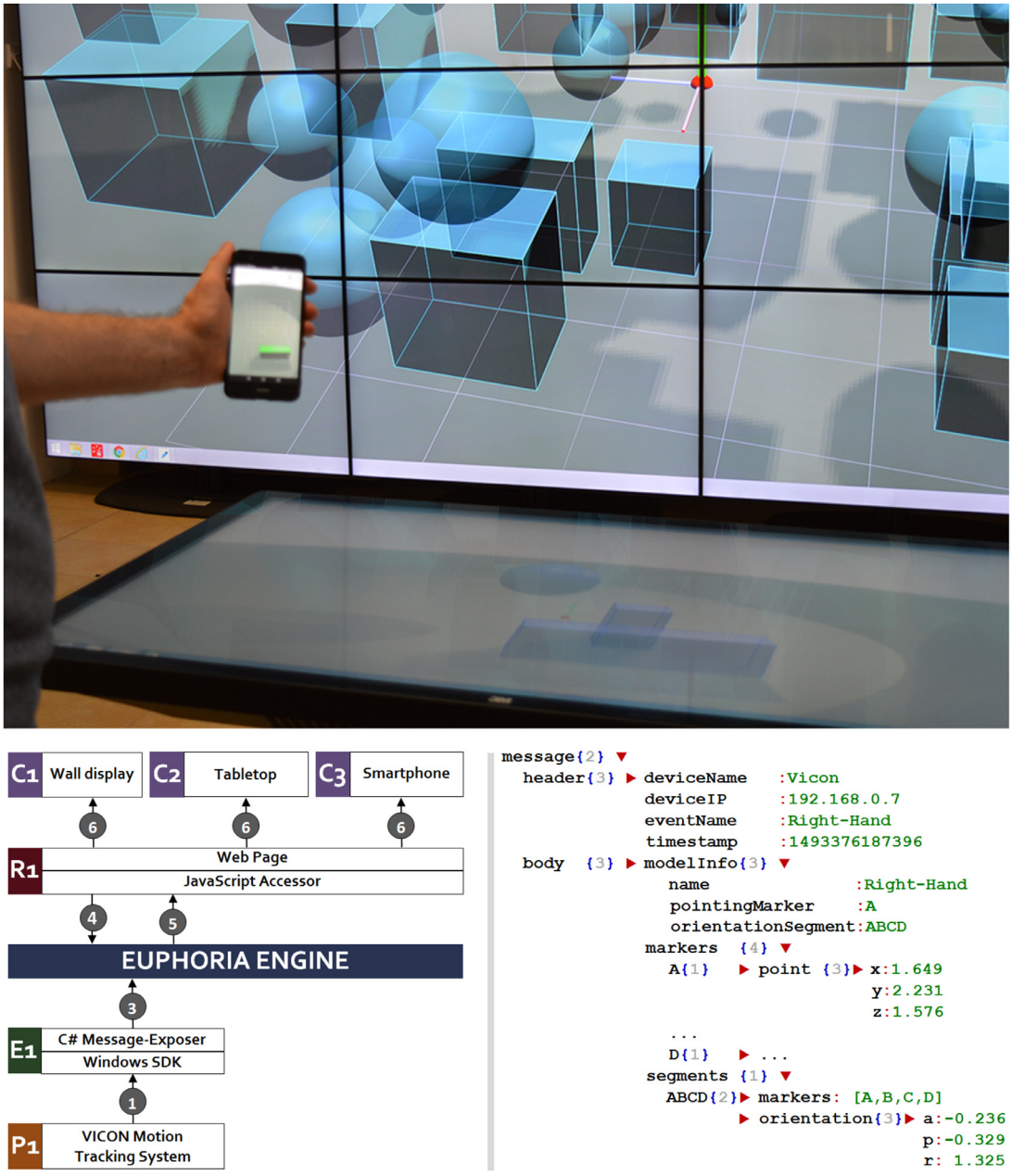


Fig. 7. EUPHORIA applied to implement a multi-device scenario to access content in a smart environment: a wall display, an interactive tabletop, and a smartphone (top image) implement consumers in the EUPHORIA architecture (bottom left) that process motion tracking events encoded as JSON messages (bottom right).

output devices were employed to display the digital content pointed to by users: (1) a Samsung UE55D Wall Display with an active area of 13m^2 and 8K Ultra-HD resolution, (2) a 46-inch Idem Platform Full-HD interactive table, and (3) a 7-inch Android tablet with 800 × 1280 screen resolution; see Fig. 7, top. Three digital layers were created on top of the physical space by associating multimedia content (in this specific implementation, links to YouTube videos) with physical regions in mid-air shaped as cuboids; see Fig. 7, top for a visual illustration. When the user enters one of the physical regions, the associated content is rendered on all devices, according to the capabilities of each device: from the personal, mobile device to the large ambient display, screen size, display quality, and computing resources increase. Therefore, the presentation of content on each device was adapted to best match the display and computing capabilities of each device. Using EUPHORIA made this task simple, where each device was implemented as a CONSUMER that required content to be delivered at a specific pixel resolution or for a specific complexity of the digital environment, *i.e.*, fewer graphical objects were displayed on devices with lesser computing capabilities.

Fig. 7, bottom illustrates the instantiation of EUPHORIA for this specific implementation as well as examples of event messages. The location of the user in the physical environment is detected, tracked, and reported by the Vicon system using the Vicon Datastream SDK (dataflow ❶ in Fig. 7), while a custom Message-Expose-Layer creates the messages with the information about the Vicon event. The same layer shares the messages with the `node.js` server (dataflow ❸ in Fig. 7) via HTTP POST requests. To receive those messages, the large display, the interactive tabletop, and the tablet are instantiated as CONSUMERS and each run a web page implementing `WebSockets` (dataflows ❹ and ❺ in Fig. 7). When the user enters a predefined region of the physical space, the digital content associated to that region is rendered differently by each device. Fig. 7, top shows a visual representation of the physical regions as cuboids superimposed on a schematic representation of the

physical space. Rendering of the graphical objects was implemented using the `three.js` 3-D library for `JavaScript`.[3] Different versions of the web page are loaded on each device, depending on the corresponding pixel resolution, screen size, and the computing resources available, *i.e.*, the large display offers a full view of the entire digital environment, while smaller subsets of objects are shown on the tabletop and the mobile device, respectively; see Fig. 7, top.

5.3. A Wearable User interface for social notifications in an in-Vehicle smart environment

We continue our discussion with another use case for EUPHORIA, this time for a specific smart environment, *i.e.*, the interior of a connected car [99,100], to show the usefulness and adaptability of EUPHORIA to a variety of physical settings and contexts of use. For the purpose of this use case, we are interested in the in-vehicle user interface [101] and an application scenario for delivering the driver with recent notifications from their social networks, such as incoming text messages to their mobile phone or recent tweets. Recently, there has been a growing interest in designing gesture user interfaces to interact with the infotainment system of modern, connected cars [102–104], so we follow this trend in our use case as well. The driver's smartphone alerts EUPHORIA about incoming calls, messages, or updates from social networks, such as the latest tweets [105]. In the case of such events occurring, feedback is provided in the form of audio messages read to the driver by the In-Vehicle Infotainment System (IVIS) of the car or as subtle vibrations on the wrist; see Fig. 8, top for a visual illustration of the in-vehicle environment. The smartwatch also acts as an input device that can detect taps, touches, and motion gestures performed with the wrist or the hand in mid-air.

Fig. 8, bottom represents the instantiation of EUPHORIA for this specific implementation as well as examples of event messages. PRODUCERS are represented by the smartwatch that reads user input and by the applications running on the smartphone that notify EUPHORIA about incoming phone calls, messages, or tweets. The IVIS system (simulated with a software application running on a laptop for this implementation) instantiates a CONSUMER that receives messages from PRODUCERS and delivers the messages to other subscribing CONSUMERS. A particularity of this scenario is that the smartwatch acts both as a PRODUCER and as a CONSUMER since it detects user input, but also notifies the driver about incoming alerts via vibrotactile feedback. The double role of the smartwatch is readily handled in EUPHORIA due to the separation of layers and the modularity of our architecture.

6. Technical evaluation

We conducted an evaluation of the technical performance of EUPHORIA by quantifying the effect of several key factors (*i.e.*, message size, device type, and the complexity of the smart environment in terms of the number of producers and consumers) on the time performance of EUPHORIA for creating, transmitting, and processing events. We also use the response time measurements to estimate the upper bound of the workload that EUPHORIA can effectively process under the constraints of real-time operation, which we define as instantaneously-perceived response.

6.1. Design

We are primarily interested in the response time for processing events in EUPHORIA, which we evaluate using the REQUEST-RESPONSE-TIME dependent variable defined as the average time, expressed in milliseconds, required for an event to be processed by the EUPHORIA architecture, from its creation by a PRODUCER to its consumption by a CONSUMER. Our technical evaluation procedure was a within-subject, repeated-measures design with the following three independent variables:

1. MESSAGE-SIZE, interval variable with five conditions representing message sizes ranging from 64 bytes to 16 Kbytes in a geometric progression with common ratio 4, *i.e.*, 64, 256, 1024, 4096, and 16,384 bytes, respectively. MESSAGE-SIZE represents the amount of information contained by an event created, transmitted, and processed by EUPHORIA (see Fig. 4 from the previous section for examples of messages) and is used to evaluate the scalability property Q1. Our null hypotheses (denoted with H_{i0} in the following, where i is the hypothesis number) state that the size of the message does not affect the request-response time (H_{10}) and, consequently, there is no relationship between the two variables (H_{20}). The alternative hypotheses (denoted with H_{ia} in the following) state that larger messages will require more time to process (H_{1a}) and the dependency between response time and message size is linear (H_{2a}):

   H_{10}: The message size does not affect the request response time of processing messages in EUPHORIA.
   H_{1a}: Larger event messages will be processed in EUPHORIA with longer request-response times.
   H_{20}: There is no relationship between the size of an event message and the time required to process the message in EUPHORIA.
   H_{2a}: A linear dependency exists between the size of an event message and the time required to process it.

2. DEVICE-TYPE, ordinal variable with three conditions representing devices with *low-end, midrange*, and *high-end* computing capability; see the "Apparatus" subsection next for details on representative devices for each category. We control DEVICE-TYPE in order to understand the effect of computing capabilities on REQUEST-RESPONSE-TIME with the following hypotheses:

   H_{30}: There is no relationship between the computing resources available to a PRODUCER or CONSUMER device and the request-response time.
   H_{3a}: The more computing resources available to a PRODUCER/CONSUMER, the shorter the response time will be.

3. ENVIRONMENT-COMPLEXITY, interval variable, representing the number of CONSUMERS running requests and receiving responses in EUPHORIA at the same time. To understand thoroughly the implications of the number of consumers on the real-time performance of EUPHORIA, we varied the complexity of the environment from 1 (one consumer) to 101 with values in an arithmetic progression with the common difference 10, *i.e.*, 1, 11, 21, ..., 91, and 101 consumers. We use ENVIRONMENT-COMPLEXITY to evaluate the scalability property Q1 intended for EUPHORIA. Our hypotheses for the ENVIRONMENT-COMPLEXITY independent variable are:

   H_{40}: The number of consumers simultaneously connected to EUPHORIA does not affect the average request-response times.
   H_{4a}: Environments with a larger number of consumers will cause longer request-response times in EUPHORIA.
   H_{50}: There is no relationship between the number of consumers and the average request-response time.
   H_{5a}: The number of consumers in the smart environment and the average response time follow a linear relationship.

6.2. Methodology

We varied the MESSAGE-SIZE and content of each message and randomized the order of DEVICE-TYPE and of the ENVIRONMENT-COMPLEXITY conditions. As a default benchmarking scenario for our evaluation, every message generated by each PRODUCER was broadcasted by EUPHORIA to all the CONSUMERS, which enabled us to measure request-response times for conditions of maximum network load.

[3] three.js - JavaScript 3D library, https://threejs.org/

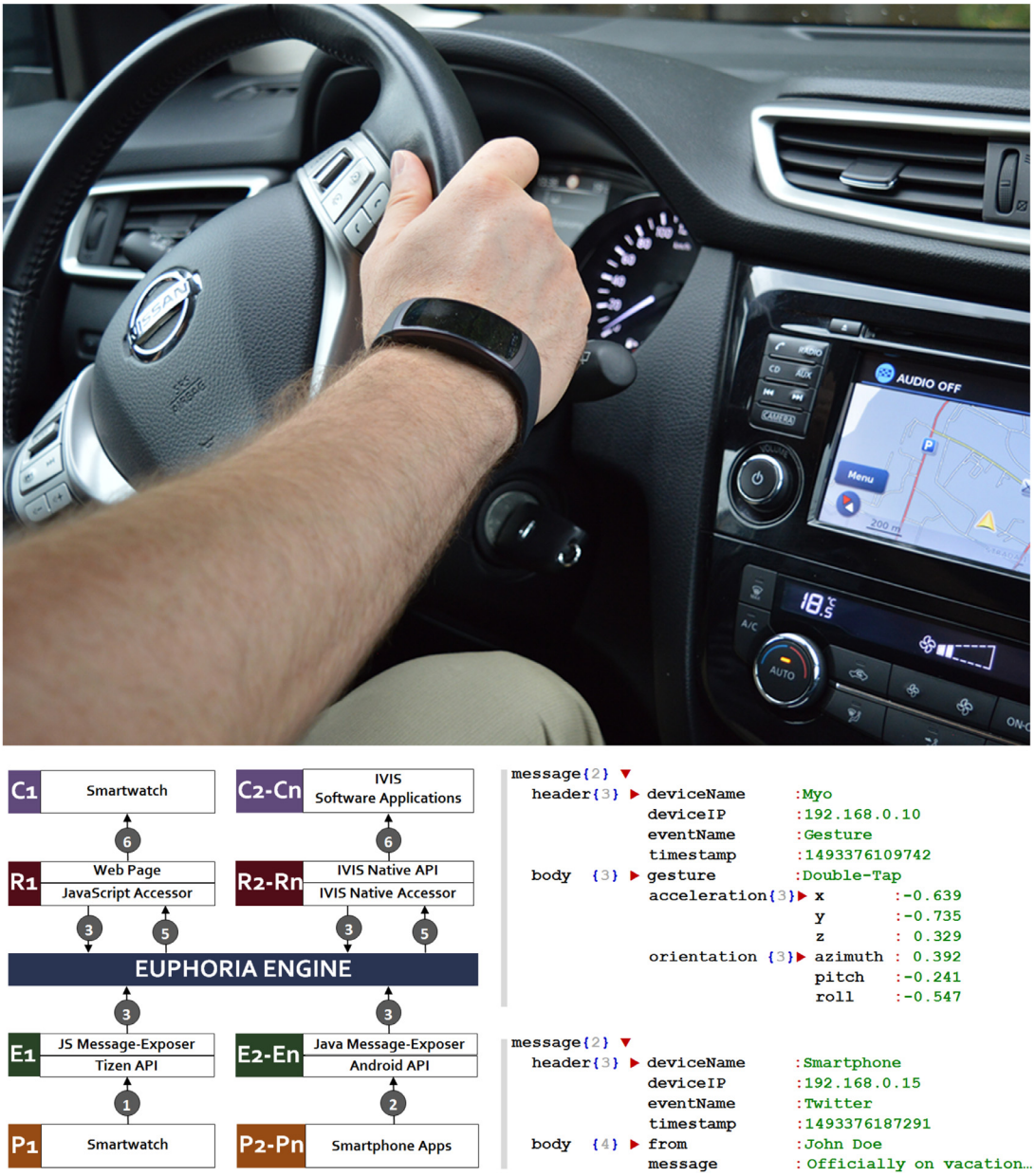


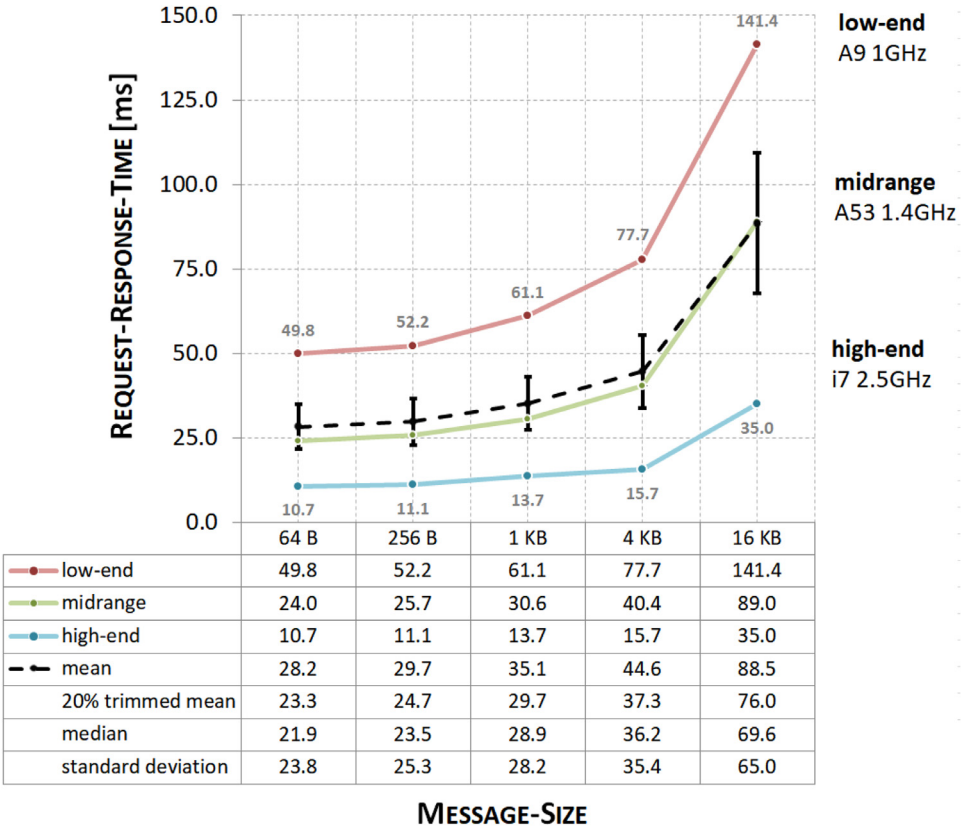
Fig. 8. EUPHORIA applied to implement a wearable user interface for social notifications in an in-vehicle smart environment: a smartwatch (top) acts as both producer and consumer in the EUPHORIA architecture (bottom left) to process events from smartphone applications, but also to enter commands (bottom right).

Because we noticed that response time measurements were slightly different on different trials, we decided to perform multiple, repeated measurements and to average request-response times over such multiple runs. We empirically ascertained that average request-response times remained stable for a number of repetitions of at least 1,000. Informed by these results, we ran 1000 repeated trials for each combination of MESSAGE-SIZE × DEVICE-TYPE × ENVIRONMENT-COMPLEXITY, resulting in $5 \times 3 \times 11 \times 1000 = 165{,}000$ trials. However, in the following analysis, we aggregated the request-response times to avoid the effects of pseudoreplication.

6.3. Apparatus

The ENGINE module of our implementation of the EUPHORIA architecture ran on a 64-bit Intel Core i7-4510U CPU with 2.60 GHz, 8 GB RAM, and Windows 7. PRODUCERS, EMITTERS, RECEIVERS, and CONSUMERS were implemented on the following mobile devices, representing the three conditions of the DEVICE-TYPE independent variable:

1. Samsung Galaxy S III, a smartphone with 32-bit dual-core Cortex A9 CPU @1GHz and 1GB RAM, running Android 4.1. This device represents our *low-end* CPU condition due to the fact that 1GHz CPUs are now common on many wearables, such as smartwatches (*e.g.*, Samsung's Gear Fit 2), while the A9 ARM CPU has been superseded by more advanced CPU architectures, such as the ones discussed next.
2. Samsung Galaxy Tab 4, a tablet with 64-bit quad-core Cortex A53 CPU @1.4GHz, 1.5GB RAM, running Android 4.4. This device is representative for our *midrange* condition because, when compared to the A9 architecture, A53 has more CPU cores, higher CPU frequency, support for 64-bit applications, hardware-assisted visualization, dynamic frequency scaling, and an enhanced Floating-Point Unit. More RAM memory and an improved operating system imply better performance.
3. Acer Aspire V15 Nitro, a notebook with 64-bit quad-core i7-4710 CPU @2.50GHz, 12 GB RAM, running Windows 10. Compared to A9 and A53, the i7 CPU architecture relies on an advanced tri-gate



	64 B	256 B	1 KB	4 KB	16 KB
low-end	49.8	52.2	61.1	77.7	141.4
midrange	24.0	25.7	30.6	40.4	89.0
high-end	10.7	11.1	13.7	15.7	35.0
mean	28.2	29.7	35.1	44.6	88.5
20% trimmed mean	23.3	24.7	29.7	37.3	76.0
median	21.9	23.5	28.9	36.2	69.6
standard deviation	23.8	25.3	28.2	35.4	65.0

Fig. 9. Mean REQUEST-RESPONSE-TIME measurements, in milliseconds, function of the MESSAGE-SIZE and DEVICE-TYPE independent variables. *Notes:* the doted line shows the mean response times for all devices; the horizontal axis is logarithmic with base 4; error bars show 95% CIs for the mean response time (all devices considered).

transistor technology that allows low power consumption and high processing speed. This CPU also offers hyper-threading technology and dedicated instruction sets, such as FMA (Fused Multiply-Add) and AVX (Advanced Vector Extensions). Therefore, this device represents our choice for the *high-end* CPU condition.

EUPHORIA and the above devices representing PRODUCERS and CONSUMERS were connected through a 300 Mbps Wireless N Gigabit network.

6.4. Results

Fig. 9 illustrates the effect of MESSAGE-SIZE on the REQUEST-RESPONSE-TIME performance of EUPHORIA for each condition of the DEVICE-TYPE independent variable. On average, REQUEST-RESPONSE-TIMES varied between 11 ms (for the smallest message of just 64 bytes received by consumers running on the high-end device) and 141 ms (for messages of 16 KBytes broadcasted to consumers running on the low-end device). These results allow us to reject hypothesis H_{10} and to accept H_{1a}: larger messages are processed with longer request-response times. The effect of ENVIRONMENT-COMPLEXITY on REQUEST-RESPONSE-TIME is illustrated in Fig. 10 for each DEVICE-TYPE. On average, request-response times varied from 9 ms (for the condition with one consumer running on the high-end device) and 113 ms (when messages were broadcasted to 101 consumers running on the low-end device), with an average response time of 47.4 ms ($SD = 45.5$ ms). These results enable us to reject hypothesis H_{40} and to accept hypothesis H_{4a}. The trends from Figs. 9 and 10 also suggest to reject hypothesis H_{30} and accept H_{3a}: devices with more computing resources process messages faster in terms of the average request-response time.

Our analysis also showed significant linear relationships (at $p < .001$) between REQUEST-RESPONSE-TIME (denoted with T in the equations below and expressed in milliseconds) and MESSAGE-SIZE (denoted with *MS*) and ENVIRONMENT-COMPLEXITY (*EC*) for each device category ($R^2 > .836$), but also overall ($R^2 = .874$), as follows:

$$\text{low-end devices: } T = 0.005 \cdot MS + 0.749 \cdot EC + 14.374 \ [ms] \quad (R^2 = 0.873, F = 185.900, p < 0.001)$$

$$\text{midrange devices: } T = 0.004 \cdot MS + 0.437 \cdot EC + 2.545 \ [ms] \quad (R^2 = 0.836, F = 138.604, p < 0.001)$$

$$\text{high-end devices: } T = 0.001 \cdot MS + 0.152 \cdot EC + 3.145 \ [ms] \quad (R^2 = 0.953, F = 550.153, p < 0.001)$$

$$\text{all devices: } T = 0.004 \cdot MS + 0.446 \cdot EC + 6.689 \ [ms] \quad (R^2 = 0.874, F = 188.946, p < 0.001)$$

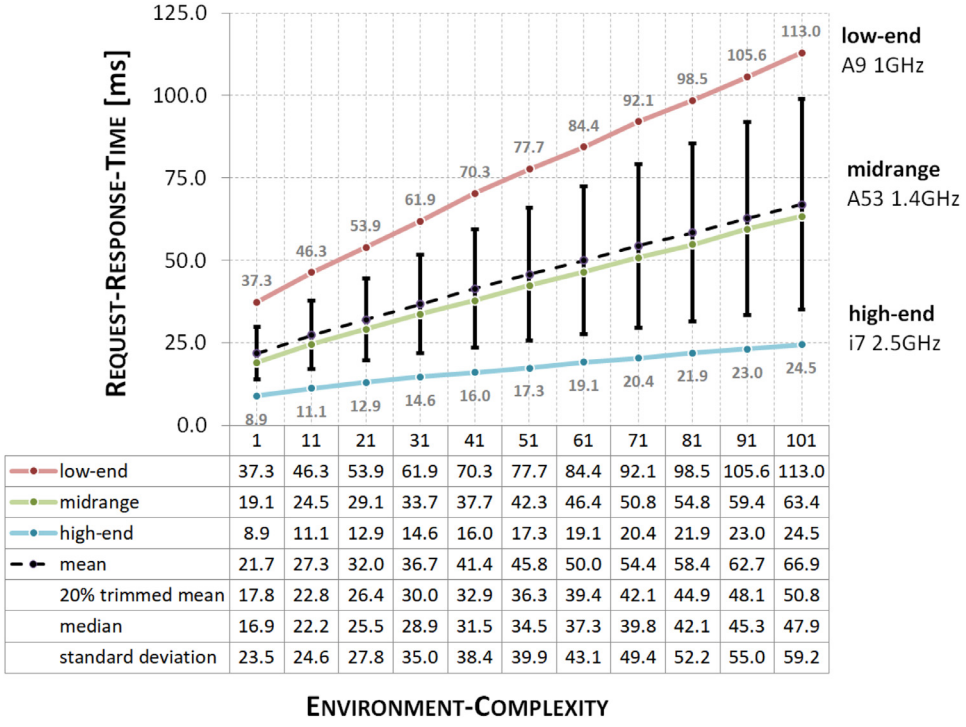
These findings support rejection of hypotheses H_{20} and H_{50} (all relationships are statistically significant at $p < .001$) and, therefore, enable us to accept the alternative hypotheses H_{2a} and H_{5a}, respectively. Moreover, we observed that the variation in request-response times (expressed using the standard deviation, σ_T, in milliseconds) also increases function of MESSAGE-SIZE (see Fig. 10) and ENVIRONMENT-COMPLEXITY (Fig. 9). We verified this observation with a second regression analysis, which showed a significant fit ($p < .001$) for each device type ($R^2 > .235$), but also overall ($R^2 = .838$), as follows:

$$\text{low-end devices: } \sigma_T = 0.001 \cdot MS + 0.031 \cdot EC + 21.600 \ [ms] \quad (R^2 = 0.412, F = 19.912, p < 0.001)$$

$$\text{midrange devices: } \sigma_T = 0.001 \cdot MS + 0.039 \cdot EC + 9.327 \ [ms] \quad (R^2 = 0.407, F = 19.497, p < 0.001)$$

$$\text{high-end devices: } \sigma_T = 0.001 \cdot MS + 0.003 \cdot EC + 7.138 \ [ms] \quad (R^2 = 0.235, F = 9.272, p < 0.001)$$

$$\text{all devices: } \sigma_T = 0.002 \cdot MS + 0.301 \cdot EC + 5.790 \ [ms] \quad (R^2 = 0.838, F = 140.967, p < 0.001)$$



	1	11	21	31	41	51	61	71	81	91	101
low-end	37.3	46.3	53.9	61.9	70.3	77.7	84.4	92.1	98.5	105.6	113.0
midrange	19.1	24.5	29.1	33.7	37.7	42.3	46.4	50.8	54.8	59.4	63.4
high-end	8.9	11.1	12.9	14.6	16.0	17.3	19.1	20.4	21.9	23.0	24.5
mean	21.7	27.3	32.0	36.7	41.4	45.8	50.0	54.4	58.4	62.7	66.9
20% trimmed mean	17.8	22.8	26.4	30.0	32.9	36.3	39.4	42.1	44.9	48.1	50.8
median	16.9	22.2	25.5	28.9	31.5	34.5	37.3	39.8	42.1	45.3	47.9
standard deviation	23.5	24.6	27.8	35.0	38.4	39.9	43.1	49.4	52.2	55.0	59.2

Fig. 10. Mean REQUEST-RESPONSE-TIME measurements, in milliseconds, function of the MESSAGE-SIZE and DEVICE-TYPE independent variables. *Notes:* the doted line shows the mean request-response time for all devices; error bars show 95% CIs for the mean time responses (all devices considered).

7. Discussion

Our empirical results show that EUPHORIA executes with an average request-response time that can be perceived as instantaneous response (below 150 ms) under various conditions represented by different message sizes and number of consumers requesting event notifications simultaneously. In this section, we concentrate on aspects regarding the internal and external validity of our evaluation and point to future work in this direction, present the open-source availability of EUPHORIA, and also provide a comparative analysis of EUPHORIA with respect to other software architectures from the perspective of our quality (Q1-Q4) and contextual properties (C1-C4).

7.1. Internal and external validity of the technical evaluation

We present in the following a critical discussion of the main threats to the validity, both internal and external, of our technical evaluation, according to the recommendations from Wohlin *et al.* [106]. We hope that this discussion will be useful to readers, who may wish to replicate our findings, but also to evaluate EUPHORIA on other devices and configurations, not considered in our evaluation.

Internal validity. We evaluated EUPHORIA in 165 distinct configurations representing all possible combinations of the MESSAGE-SIZE, DEVICE-TYPE, and ENVIRONMENT-COMPLEXITY independent variables. To avoid the *history* and *maturation* threats [106], we deactivated the automatic updates on all the devices running producers and consumers during the technical evaluation procedure, so that all trials ran on the same software configuration. Moreover, we randomized the order of the trials and conducted repeated measurements of the REQUEST-RESPONSE-TIME dependent variable with a total number of 1000 repetitions for each combination of MESSAGE-SIZE, DEVICE-TYPE, and ENVIRONMENT-COMPLEXITY. This way, we were able to measure precisely the succession of events in time and, thus, avoid the threat of *ambiguous temporal precedence* [106]. The same software and hardware configuration for EUPHORIA was assured throughout the entire technical evaluation to avoid the *instrumentality* threat [106]. Moreover, the large number of repetitions was adopted to compensate for unpredictable influences. Although we wanted to be as objective as possible, our evaluation may have been subjected to the *selection bias* threat [106] regarding the DEVICE-TYPE variable, for which we hand-picked three representative devices that, as of the year 2018, reflect well several segments of the computing market. Of course, evaluating the average request-response time of EUPHORIA on more devices, including devices with other computing characteristics, is advisable and we recommend such explorations for future work. For example, addressing tiny devices, such as smartwatches or smart glasses, represents an interesting direction to report the performance of EUPHORIA in the context of wearable computing.

External validity. We took special care to present to readers all the details required for a good *replication* [106] of our empirical results, including technical details regarding the platforms, devices, software, and network infrastructure that we used to implement and evaluate EUPHORIA. The *generalizability* threat [106] was addressed with multiple configurations representing combinations of the MESSAGE-SIZE × DEVICE-TYPE × ENVIRONMENT-COMPLEXITY independent variables. Our empirical results showed that EUPHORIA processed messages with a temporal performance that can be perceived as instantaneous response by users, *i.e.*, under 150 ms average request-response time for the slowest device (A9 CPU) that processed the largest message (64 Kbytes) in the highest density condition (101 consumers). We did not evaluate EUPHORIA against critical, real-time requirements, because EUPHORIA was not intended to act as a safety-critical system. Instead, the main objective of our evaluation was to prove that it is possible for heterogeneous devices to exchange data in a simple and efficient manner in EUPHORIA. Nevertheless, continuing work in this direction is recommended, such as addressing other devices and configurations, an outcome that we expect from the community as a direct consequence of releasing our software architecture in the public domain; see the next section.

7.2. EUPHORIA as open-source software

We deliver EUPHORIA for free for research purposes at the permanent web address **http://www.eed.usv.ro/mintviz/resources/Euphoria** along with a live demonstration, sandbox, and examples of messages. In doing this, we hope to foster advances and developments in new software architecture initiatives for our increasingly complex smart environments, but also to readily support implementations of novel interactive

Table 1
Comparison of previous architectures and EUPHORIA in terms of the four quality and four contextual properties.

	Adaptability Q1	Modularity Q2	Flexibility Q3	Interoperability Q4	Web based C1	Open source C2	Smart oriented C3	Java Script C4
UbiComp Middleware [75]	●	◑	◕	●	◑	◑	●	◔
MAPS [76]	◑	●	◕	●	◕	◑	◑	◔
SPINE [77]	◕	◕	◕	●	◑	●	◑	◔
Proximity Toolkit [82]	◕	●	◕	●	◕	●	●	◕
BMF-WSANs [78]	●	◕	◕	●	◑	◕	◕	◑
BodyCloud [17]	◕	◑	◕	◕	●	◑	●	◑
XDKinect [23]	◕	●	◑	◕	●	◕	◕	◔
FiWare [79]	●	●	◑	●	●	◕	◕	◑
Proxemic-Aware Controls [81]	◕	◑	◑	◕	◕	◔	◕	◔
Vi-Smart [24]	◕	◕	◕	◕	●	◑	●	◕
Mobile@Old [85]	◕	◑	◑	◕	◑	◑	◕	◔
GS-GPE [12]	◑	●	◑	◕	●	◕	●	◑
Overall (prior work)	◕	◕	◑	◕	◕	◕	◕	◑
Euphoria	●	●	◕	●	●	●	◕	◕

systems and applications for smart environments of all kinds. Since multiple, potentially heterogeneous, input/output devices can be integrated in EUPHORIA, researchers and practitioners are prompted to design, invent, and test new interaction modalities.

7.3. Comparison to related platforms and architectures

Table 1 presents a comparison between EUPHORIA and related platforms and software architectures for smart environments discussed in this paper. For this comparison, we relied on the four quality properties (Q1 to Q4) and the four contextual properties (C1 to C4) as differentiating factors to highlight the unique features of EUPHORIA.

Each entry from Table 1 is depicted using the visual formalism of Harvey's ideograms for qualitative information, as follows: a ball with a single quarter denotes that the criteria is qualitatively fulfilled by at least one feature, a half ball denotes that the criterion is qualitatively fulfilled by more than one feature, a three-quarter ball denotes that the criterion is fulfilled by many features that can be combined, while a full ball denotes total fulfillment. We performed the comparative analysis based solely on the descriptions available from the literature without any quantitative measurements, which would have required benchmarking systems for architectures inconsistent with each other. For example, the UbiComp Middleware [75] offers support for creating smart objects as a bridge between the physical and the digital world (C3) and it does not require internal changes to adapt to various smart environments (Q1, Q4); however, it is neither web-based (C1) or using JavaScript (C4), although it can integrate web architectures. The MAPS platform [76] is a framework for building agent-oriented WSNs (Wireless Sensor Networks) applications. Due to the use of agents, the architecture is decoupled (Q2), while the client-server model offers high interoperability (Q4). However, MAPS needs adaptation to support interactions in a smart environment (C3). SPINE [77] is a domain specific architecture for rapid prototyping of WBSNs (Wireless Body Sensor Networks) that is open source (C2), while the Java APIs can be easily ported across various devices (Q4); however, SPINE was not primarily designed for the web (C1, C4). The Proximity toolkit [82] is smart environment oriented (C3), open source (C2), highly modularized due to the plug-in approach (Q2), and relies on the client-server paradigm (C1). Proxemic-Aware Controls [84] are virtual objects representing dynamically created interfaces between users and digital appliances within a smart environment (C3) that enable discovery of interactive appliances (Q1) and communication using standardized protocols (Q4). BMF-WSANs [78] is a framework for handling networks of wireless sensors and actuators within a building. The web integration is limited to reporting energy performance over the Internet, but can be extended to support other functionalities (C1). BodyCloud [17] is a multi-layered architecture for Body Sensor Networks applications that ensures data transfer across different standards (Q4), plug and play devices (Q1, Q3), and integration of heterogeneous sensors (C3). XDKinect [23] is a client-server (C1) framework for whole-body interaction that supports integration with smart spaces (C3), but is specific to the Kinect sensor (Q3). FiWare [79] is a management solution for cloud federations (C1) built on top of the OpenStack platform (Q1, Q2) that follows the specifications of Open Cloud Computing Interface for high interoperability (Q4). Mobile@Old [85] is a platform for assisting elderly people to perform physical exercises, incorporating a physical activity trainer and a module for monitoring vital signs that can be easily integrated within a smart environment (C3), while the physical training exercises are adapted to the medical history of the user (Q1). Vi-Smart [24] is a system for collaborative ambient intelligence (C3) for people with acquired brain injury that stores specific configurations for each patient (Q1, Q3). GS-GPE [12] implements personalized gesture recognition based on user identification for smart homes (C3), but is limited to gesture input only (Q1, Q3).

7.4. The quality attributes of EUPHORIA

In our evaluation, we were primarily interested in the technical performance of EUPHORIA to characterize and understand quantitative

relationships between various involved variables, such as the effects of MESSAGE-SIZE and ENVIRONMENT-COMPLEXITY on response times. However, the quality properties envisioned for EUPHORIA are strongly connected with the empirical results obtained during the technical evaluation. For example, the quality property Q1 (adaptability/scalability) is directly linked to both the number and the type of devices exchanging messages via EUPHORIA as well as with the size of those messages. From this perspective, our technical results suggest *data-oriented scalability* (see the effect of MESSAGE-SIZE) and *environment-oriented scalability* (the effects of DEVICE-TYPE and ENVIRONMENT-COMPLEXITY) as adaptability characteristics of EUPHORIA to changes occurring in the context of use. The modularity property (Q2) results from the layered architecture design as well as from the loose coupling communication (*i.e.*, `JSON` messages sent through `http` requests and `WebSockets`): changes originating in one module determine no changes in the adjacent modules as long as the interfacing stays the same. The DEVICE-TYPE variable is also connected with the flexibility property (Q3) of EUPHORIA, since it addresses the capability of EUPHORIA to handle various hardware and software configurations. Moreover, once the EMITTER and the RECEIVER are implemented for any particular device, they can be readily reused for similar devices (*i.e.*, having the same operating system and message structure) according to the quality property Q3. Also, due to standardized communication, different devices can interact as long as they implement common web-based technology (quality property Q4). Therefore, EUPHORIA can easily communicate with other systems or with other replicas of itself and, thus, create more complex architectures. EUPHORIA also complies with the four contextual properties defined in its requirements: the use of web standards (C1), availability as an open source platform (C2), its specific design for smart spaces (C3), and core modules written in JavaScript (C4). The comparative analysis between EUPHORIA and previous software architectures suggests that EUPHORIA complies more with the aforementioned requirements than any prior work.

8. Conclusion

We introduced EUPHORIA, an asynchronous event-driven software architecture for supporting adaptable and flexible interactions across heterogeneous input/output devices in smart environments. Our technical evaluation showed that EUPHORIA delivers average response times under 150 ms for applications involving large messages exchanged between a large number of producers and consumers. Future interesting research will look at expanding these results to more complex and demanding scenarios, potentially involving hundreds or thousands of concurrent users, such as for mass-computer interaction [61]. Also, further evaluations of the technical performance of EUPHORIA on other types of devices, such as wearable gadgets with less computing power, and in other smart environments, as well as further evaluation of the quality properties in various contexts of application supported by EUPHORIA, such as natural interaction, are envisaged. By introducing EUPHORIA that enables simple integration of applications, devices, APIs, and source code independently of their software/hardware platforms, operating systems, and native programming languages, we hope to foster advances and developments in new software architecture initiatives for our increasingly complex smart environments, but also to readily support implementations of novel interactive systems and applications for smart environments of all kinds.

Acknowledgement

This work was supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI-UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-0917, contract no. 21PCCDI/2018, within PNCDI III, project P_2, *"Efficient communications based on smart devices for in-car augmented reality interactive applications."* Research was conducted in the Machine Intelligence and Information Visualization Lab (MintViz) of the MANSiD Research Center. The infrastructure was provided by the University of Suceava and was partially supported from the project *"Integrated center for research, development and innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for fabrication and control"*, No. 671/09.04.2015, Sectoral Operational Program for Increase of the Economic Competitiveness, co-funded from the European Regional Development Fund.

References

[1] S. Poslad, Ubiquitous Computing: Smart Devices, Environments and Interactions, John Wiley & Sons, West Sussex, UK, 2011, doi:10.1002/9780470779446.
[2] N.F. Raun, Smart environment using internet of things(iots) - a review, in: 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016, pp. 1–6, doi:10.1109/IEMCON.2016.7746313.
[3] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J. Vanderdonckt, A unifying reference framework for multi-target user interfaces, Interact. Comput. 15 (3) (2003) 289–308, doi:10.1016/S0953-5438(03)00010-9.
[4] A.D. Wilson, H. Benko, Combining multiple depth cameras and projectors for interactions on, above and between surfaces, in: Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology, in: UIST '10, ACM, New York, NY, USA, 2010, pp. 273–282, doi:10.1145/1866029.1866073.
[5] R.-D. Vatavu, Nomadic gestures: a technique for reusing gesture commands for frequent ambient interactions, J. Ambient Intell. Smart. Environ. 4 (2) (2012) 79–93, doi:10.3233/AIS-2012-0137.
[6] R.-D. Vatavu, Smart-pockets: body-deictic gestures for fast access to personal data during ambient interactions, Int. J. HumanComput. Stud. 103 (2017) 1–21, doi:10.1016/j.ijhcs.2017.01.005.
[7] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): a vision, architectural elements, and future directions, Future Gen. Comput. Syst. 29 (7) (2013) 1645–1660, doi:10.1016/j.future.2013.01.010.
[8] A. Greenfield, Everyware: The Dawning Age of Ubiquitous Computing, Peachpit Press, Berkeley, CA, USA, 2006.
[9] P. Friess, Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, River Publishers, Aalborg, Denmark, 2013.
[10] O.-A. Schipor, W. Wu, W.-T. Tsai, R.-D. Vatavu, Software architecture design for spatially-indexed media in smart environments, Adv. Electr. Comput. Eng. 17 (2) (2017) 17–22, doi:10.4316/AECE.2017.02003.
[11] R.-D. Vatavu, C.-M. Chera, W.-T. Tsai, Gesture profile for web services: An event-driven architecture to support gestural interfaces for smart environments, in: International Joint Conference on Ambient Intelligence, Springer, 2012, pp. 161–176, doi:10.1007/978-3-642-34898-3_11.
[12] Y. Lou, W. Wu, R.-D. Vatavu, W.-T. Tsai, Personalized gesture interactions for cyber-physical smart-home environments, Sci. China Inf. Sci. 60 (7) (2017) 072104, doi:10.1007/s11432-015-1014-7.
[13] S. Gustafson, D. Bierwirth, P. Baudisch, Imaginary interfaces: spatial interaction with empty hands and without visual feedback, in: Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology, in: UIST '10, ACM, New York, NY, USA, 2010, pp. 3–12, doi:10.1145/1866029.1866033.
[14] B.-F. Gheran, J. Vanderdonckt, R.-D. Vatavu, Gestures for smart rings: empirical results, insights, and design implications, in: Proceedings of the 2018 Designing Interactive Systems Conference, in: DIS '18, ACM, New York, NY, USA, 2018, pp. 623–635, doi:10.1145/3196709.3196741.
[15] ThalmicLabs, Myo gesture control armband—wearable technology by thalmic labs, 2018 https://www.myo.com/.
[16] G. Bailly, J. Müller, M. Rohs, D. Wigdor, S. Kratz, Shoesense: a new perspective on gestural interaction and wearable applications, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, in: CHI '12, ACM, New York, NY, USA, 2012, pp. 1239–1248, doi:10.1145/2207676.2208576.
[17] G. Fortino, D. Parisi, V. Pirrone, G. Di Fatta, Bodycloud: a saas approach for community body sensor networks, Future Gener. Comput. Syst. 35 (2014) 62–79, doi:10.1016/j.future.2013.12.015.
[18] P.-V. Cioată, R.-D. Vatavu, In tandem: exploring interactive opportunities for dual input and output on two smartwatches, in: Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion, in: IUI '18 Companion, ACM, New York, NY, USA, 2018, pp. 60:1–60:2, doi:10.1145/3180308.3180369.
[19] X.A. Chen, T. Grossman, D.J. Wigdor, G. Fitzmaurice, Duet: exploring joint interactions on a smart phone and a smart watch, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, in: CHI '14, ACM, New York, NY, USA, 2014, pp. 159–168, doi:10.1145/2556288.2556955.
[20] A. Esteves, E. Velloso, A. Bulling, H. Gellersen, Orbits: gaze interaction for smart watches using smooth pursuit eye movements, in: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, in: UIST '15, ACM, New York, NY, USA, 2015, pp. 457–466, doi:10.1145/2807442.2807499.
[21] M. Villari, A. Celesti, M. Fazio, A. Puliafito, Alljoyn lambda: an architecture for the management of smart environments in iot, in: Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on, IEEE, 2014, pp. 9–14, doi:10.1109/SMARTCOMP-W.2014.7046676.
[22] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, Middlewares for smart objects and smart environments: overview and comparison, in: Internet of Things Based on Smart Objects, Springer, 2014, pp. 1–27, doi:10.1007/978-3-319-00491-4_1.
[23] M. Nebeling, E. Teunissen, M. Husmann, M.C. Norrie, Xdkinect: development framework for cross-device interaction using kinect, in: Proceedings of the 2014

ACM SIGCHI Symposium on Engineering Interactive Computing Systems, in: EICS '14, ACM, New York, NY, USA, 2014, pp. 65–74, doi:10.1145/2607023.2607024.
[24] C. Roda, A. Rodríguez, E. Navarro, V. López-Jaquero, P. González, Towards an architecture for a scalable and collaborative ami environment, in: Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection, Springer, 2016, pp. 311–323, doi:10.1007/978-3-319-40159-1_26.
[25] R.D. Hill, Event-response systems: a technique for specifying multi-threaded dialogues, SIGCHI Bull. 18 (4) (1986) 241–248, doi:10.1145/1165387.275637.
[26] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, R. Little, Documenting Software Architectures: Views and Beyond, Pearson Education, Boston, MA, USA, 2002.
[27] D. Garland, M. Shaw, An introduction to software architecture, in: V. Ambriola, G. Tortora (Eds.), Advances in Software Engineering and Knowledge Engineering, World Scientific Publishing, Singapore, 1993, pp. 1–39 https://doi.org/10.1142/9789812798039_0001.
[28] D.E. Perry, A.L. Wolf, Foundations for the study of software architecture, ACM SIGSOFT Softw. Eng. Notes 17 (4) (1992) 40–52, doi:10.1145/141874.141884.
[29] A metamodel for the runtime architecture of an interactive system: the uims tool developers workshop, SIGCHI Bull. 24 (1) (1992) 32–37, doi:10.1145/142394.142401.
[30] P.D. Bruza, Th.P. van der Weide, The semantics of data flow diagrams, in: Proceedings of the International Conference on Management of Data, McGraw-Hill, Inc, New York, NY, USA, 1993, pp. 66–78.
[31] E. Yourdon, L.L. Constantine, Structured design: Fundamentals of a discipline of computer program and systems design, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1979.
[32] C. Moxey, M. Edwards, O. Etzion, M. Ibrahim, S. Iyer, H. Lalanne, M. Monze, M. Peters, Y. Rabinovich, G. Sharon, et al., A conceptual model for event processing systems, IBM Redguide Publication, 2010. https://www.ibm.com/developerworks/library/ws-eventprocessing/index.html
[33] K.M. Chandy, Event-driven applications: costs, benefits and design approaches, Gartner Appl. Integr. Web Serv. Summit 2006 (2006).
[34] D.L. Parnas, On the criteria to be used in decomposing systems into modules (Reprint), in: M. Broy, E. Denert (Eds.), Software Pioneers., Springer Berlin Heidelberg, 2002, pp. 411–427, doi:10.1007/978-3-642-59412-0_26.
[35] B.M. Michelson, Event-driven architecture overview, Patric. Seybold Group 2 (2006).
[36] H. Taylor, A. Yochem, L. Phillips, F. Martinez, Event-Driven Architecture: How SOA Enables the Real-Time Enterprise, 1st, Addison-Wesley Professional, 2009.
[37] D.M. Hilbert, D.F. Redmiles, Extracting usability information from user interface events, ACM Comput. Surv. (CSUR) 32 (4) (2000) 384–421, doi:10.1145/371578.371593.
[38] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, SIGARCH Comput. Archit. News 28 (5) (2000) 93–104, doi:10.1145/378995.379006.
[39] M. Ilyas, I. Mahgoub, Smart dust: sensor network applications, architecture and design, CRC Press, Boca Raton, FL, USA, 2016.
[40] W. Wang, Y. Xu, M. Khanna, A survey on the communication architectures in smart grid, Comput. Netw. 55 (15) (2011) 3604–3629, doi:10.1016/j.comnet.2011.07.010.
[41] C.-h. Yang, G. Zhabelova, C.-W. Yang, V. Vyatkin, Cosimulation environment for event-driven distributed controls of smart grid, IEEE Trans. Ind. Inf. 9 (3) (2013) 1423–1435, doi:10.1109/TII.2013.2256791.
[42] P. Shaya, P. Sander, D. Remco, G. Paul, A systematic literature review on the architecture of business process management systems, Inf. Syst. 66 (C) (2017) 43–58, doi:10.1016/j.is.2017.01.007.
[43] K. Julian, W. Benjamin, W. Dirk, P. Loos, Event-driven business process management: where are we now? a comprehensive synthesis and analysis of literature, Bus. Process Manag. J. 20 (4) (2014) 615–633, doi:10.1108/BPMJ-07-2013-0092.
[44] E. Patti, A. Acquaviva, M. Jahn, F. Pramudianto, R. Tomasi, D. Rabourdin, J. Virgone, E. Macii, Event-driven user-centric middleware for energy-efficient buildings and public spaces, IEEE Syst. J. 10 (3) (2016) 1137–1146, doi:10.1109/JSYST.2014.2302750.
[45] J. Morales, M. Garcia, Geosmart cities: Event-driven geoprocessing as enabler of smart cities, in: Smart Cities Conference (ISC2), 2015 IEEE First International, IEEE, 2015, pp. 1–6, doi:10.1109/ISC2.2015.7366207.
[46] L.A. Camuñas-Mesa, T. Serrano-Gotarredona, S.H. Ieng, R.B. Benosman, B. Linares-Barranco, On the use of orientation filters for 3d reconstruction in event-driven stereo vision, Front Neurosci. (2014), doi:10.3389/fnins.2014.00048.
[47] P.A. Merolla, J.V. Arthur, R. Alvarez-Icaza, A.S. Cassidy, J. Sawada, F. Akopyan, B.L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al., A million spiking-neuron integrated circuit with a scalable communication network and interface, Science 345 (6197) (2014) 668–673, doi:10.1126/science.1254642.
[48] L. Li, D.W. Ho, S. Xu, A distributed event-triggered scheme for discrete-time multi-agent consensus with communication delays, IET Control Theory Appl. 8 (10) (2014) 830–837, doi:10.1049/iet-cta.2013.0761.
[49] M. Weiser, R. Gold, J.S. Brown, The origins of ubiquitous computing research at parc in the late 1980s, IBM Syst. J. 38 (4) (1999) 693–696, doi:10.1147/sj.384.0693.
[50] E. Aarts, R. Harwig, M. Schuurmans, The Invisible Future: The Seamless Integration of Technology into Everyday Life, McGraw-Hill, Inc., New York, NY, USA, 2002, pp. 235–250. http://dl.acm.org/citation.cfm?id=504949.504964
[51] M. Kuniavsky, Smart Things: Ubiquitous Computing User Experience Design, Morgan Kaufmann, Burlington, MA, USA, 2010, doi:10.1016/C2009-0-20057-2.
[52] D. Cook, S.K. Das, Smart Environments: Technology, Protocols and Applications, Wiley-Interscience, 2004, doi:10.1002/047168659X.
[53] D. Korzun, I. Galov, A. Kashevnik, S. Balandin, Virtual Shared Workspace for Smart Spaces and m3-Based Case Study, in: Open Innovations Association FRUCT, Proceedings of 15th Conference of, IEEE, 2014, pp. 60–68, doi:10.1109/FRUCT.2014.6872437.
[54] R.-D. Vatavu, A. Mossel, C. Schönauer, Digital vibrons: Understanding users' perceptions of interacting with invisible, zero-weight matter, in: Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services, in: MobileHCI '16, ACM, New York, NY, USA, 2016, pp. 217–226, doi:10.1145/2935334.2935364.
[55] M.A. Rahman, M.S. Hossain, A gesture-based smart home-oriented health monitoring service for people with physical impairments, in: Proceedings of the 14th International Conference on Inclusive Smart Cities and Digital Health - Volume 9677, in: ICOST 2016, Springer-Verlag, Berlin, Heidelberg, 2016, pp. 464–476, doi:10.1007/978-3-319-39601-9_42.
[56] M.J. Deen, Information and communications technologies for elderly ubiquitous healthcare in a smart home, Pers. Ubiquitous Comput. 19 (3–4) (2015) 573–599, doi:10.1007/s00779-015-0856-x.
[57] M.-K. Le, H.-T. Chang, Y.-M. Chang, Y.-H. Hu, H.-T. Chen, An efficient multilevel healthy cloud system using spark for smart clothes, in: Computer Symposium (ICS), 2016 International, IEEE, 2016, pp. 182–186, doi:10.1109/ICS.2016.0044.
[58] A. Pfister, A.M. West, S. Bronner, J.A. Noah, Comparative abilities of microsoft kinect and vicon 3d motion capture for gait analysis, J. Med. Eng. Technol. 38 (5) (2014) 274–280, doi:10.3109/03091902.2014.909540.
[59] M. Lui, J.D. Slotta, Immersive simulations for smart classrooms: exploring evolutionary concepts in secondary science, Technol. Pedag. Educ. 23 (1) (2014) 57–80, doi:10.1080/1475939X.2013.838452.
[60] B. Brown, K. O'Hara, T. Kindberg, A. Williams, Crowd computer interaction, in: CHI '09 Extended Abstracts on Human Factors in Computing Systems, in: CHI EA '09, ACM, New York, NY, USA, 2009, pp. 4755–4758, doi: 10.1145/1520340.1520733.
[61] J.-Y.L. Lawson, J. Vanderdonckt, R.-D. Vatavu, Mass-computer interaction for thousands of users and beyond, in: Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems, in: CHI EA '18, ACM, New York, NY, USA, 2018. LBW032:1–LBW032:6 10.1145/3170427.3188465.
[62] V.G. Motti, J. Vanderdonckt, A computational framework for context-aware adaptation of user interfaces, in: Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on, IEEE, 2013, pp. 1–12, doi:10.1109/RCIS.2013.6577709.
[63] D. Korzun, On the smart spaces approach to semantic-driven design of service-oriented information systems, in: International Baltic Conference on Databases and Information Systems, Springer, 2016, pp. 181–195, doi:10.1007/978-3-319-40180-5_13.
[64] O.A. Schipor, Improving computer assisted speech therapy through speech based emotion recognition, in: Conference proceedings of eLearning and Software for Education (eLSE), Editura Univ. Carol I, 2014, pp. 101–104. http://arxiv.org/abs/1405.7796
[65] K. Bahreini, R. Nadolski, W. Westera, Towards multimodal emotion recognition in e-learning environments, Interact. Learn. Environ. 24 (3) (2016) 590–605, doi:10.1080/10494820.2014.908927.
[66] C. Ye, Y. Xia, Y. Sun, S. Wang, H. Yan, R. Mehmood, Erar: an event-driven approach for real-time activity recognition, in: Identification, Information, and Knowledge in the Internet of Things (IIKI), 2015 International Conference on, IEEE, 2015, pp. 288–293, doi:10.1109/IIKI.2015.69.
[67] D.J. Cook, N.C. Krishnan, P. Rashidi, Activity discovery and activity recognition: a new partnership, IEEE Trans. Cybern. 43 (3) (2013) 820–828, doi:10.1109/TSMCB.2012.2216873.
[68] S.S. Rautaray, A. Agrawal, Vision based hand gesture recognition for human computer interaction: a survey, Artif. Intell. Rev. 43 (1) (2015) 1–54, doi:10.1007/s10462-012-9356-9.
[69] M.R. Abid, E.M. Petriu, E. Amjadian, Dynamic sign language recognition for smart home interactive application using stochastic linear formal grammar, IEEE Trans. Instrum. Meas. 64 (3) (2015) 596–605, doi:10.1109/TIM.2014.2351331.
[70] O. Schipor, S. Pentiuc, M. Schipor, The utilization of feedback and emotion recognition in computer based speech therapy system, Elektronika ir Elektrotechnika 109 (3) (2011) 101–104, doi:10.5755/j01.eee.109.3.181.
[71] J.K. Zao, T.T. Gan, C.K. You, S.J.R. Méndez, C.E. Chung, Y. Te Wang, T. Mullen, T.P. Jung, Augmented brain computer interaction based on fog computing and linked data, in: Intelligent Environments (IE), 2014 International Conference on, IEEE, 2014, pp. 374–377, doi:10.1109/IE.2014.54.
[72] R.-D. Vatavu, Nomadic gestures: a technique for reusing gesture commands for frequent ambient interactions, J. Ambient Intell. Smart Environ. 4 (2) (2012) 79–93. http://dl.acm.org/citation.cfm?id=2350758.2350765.
[73] M. Funk, A. Sahami, N. Henze, A. Schmidt, Using a touch-sensitive wristband for text entry on smart watches, in: CHI '14 Extended Abstracts on Human Factors in Computing Systems, in: CHI EA '14, ACM, New York, NY, USA, 2014, pp. 2305–2310, doi:10.1145/2559206.2581143.
[74] W.-H. Chen, Blowatch: Blowable and hands-free interaction for smartwatches, in: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, in: CHI EA '15, ACM, New York, NY, USA, 2015, pp. 103–108, doi:10.1145/2702613.2726961.
[75] C. Goumopoulos, A. Kameas, P. Hellas, Smart objects as components of ubicomp applications, Int. J. Multimed. Ubiquit.Eng. 4 (3) (2009). http://www.sersc.org/journals/IJMUE/vol4_no3_2009/1.pdf
[76] F. Aiello, G. Fortino, R. Gravina, A. Guerrieri, A java-based agent platform for programming wireless sensor networks†, Comput. J. 54 (3) (2011) 439–454, doi:10.1093/comjnl/bxq019.

[77] F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, M. Sgroi, Spine: a domain-specific framework for rapid prototyping of wbsn applications, Softw. Pract. Exper. 41 (3) (2011) 237–265, doi:10.1002/spe.998.
[78] G. Fortino, A. Guerrieri, G.M.P. O'Hare, A. Ruzzelli, A flexible building management framework based on wireless sensor and actuator networks, J. Netw. Comput. Appl. 35 (6) (2012) 1934–1952, doi:10.1016/j.jnca.2012.07.016.
[79] T. Zahariadis, A. Papadakis, F. Alvarez, J. Gonzalez, F. Lopez, F. Facca, Y. Al-Hazmi, Fiware lab: managing resources and services in a cloud federation supporting future internet applications, in: Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on, IEEE, 2014, pp. 792–799, doi:10.1109/UCC.2014.129.
[80] F. Lab, Fiware lab, 2018. http://status.lab.fiware.org/.
[81] E. Hall, The hidden dimension (anchor books a doubleday anchor book), Anchor, 1966. http://www.worldcat.org/isbn/0385084765.
[82] N. Marquardt, R. Diaz-Marino, S. Boring, S. Greenberg, The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies, in: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, in: UIST '11, ACM, New York, NY, USA, 2011, pp. 315–326, doi:10.1145/2047196.2047238.
[83] A.P. Volpentesta, A framework for human interaction with mobiquitous services in a smart environment, Comput. Hum. Behav. 50 (C) (2015) 177–185, doi:10.1016/j.chb.2015.04.003.
[84] D. Ledo, S. Greenberg, N. Marquardt, S. Boring, Proxemic-aware controls: Designing remote controls for ubiquitous computing ecologies, in: Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, in: MobileHCI '15, ACM, New York, NY, USA, 2015, pp. 187–198, doi:10.1145/2785830.2785871.
[85] I. Mocanu, O.A. Schipor, A serious game for improving elderly mobility based on user emotional state, in: The International Scientific Conference eLearning and Software for Education, 2, " Carol I" National Defence University, 2017, p. 487. https://doi.org/10.12753/2066-026x-17-154.
[86] ISO, Iec 25000 software and system engineering–software product quality requirements and evaluation (square)–guide to square, International Organization for Standarization (2005). https://www.iso.org/standard/35683.html.
[87] F. Schneider, B. Berenbach, A literature survey on international standards for systems requirements engineering, Procedia Comput. Sci. 16 (2013) 796–805, doi:10.1016/j.procs.2013.01.083.
[88] D. Chatzopoulos, C. Bermejo, Z. Huang, P. Hui, Mobile augmented reality survey: from where we are to where we go, IEEE Access 5 (2017) 6917–6950, doi:10.1109/ACCESS.2017.2698164.
[89] V. Wang, F. Salim, P. Moskovits, The definitive guide to HTML5 websocket, 1, Springer, 2013, doi:10.1007/978-1-4302-4741-8.
[90] D. Skvorc, M. Horvat, S. Srbljic, Performance evaluation of websocket protocol for implementation of full-duplex web streams, in: Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on, IEEE, 2014, pp. 1003–1008, doi:10.1109/MIPRO.2014.6859715.
[91] W. Buxton, Lexical and pragmatic considerations of input structures, SIGGRAPH Comput. Graph. 17 (1) (1983) 31–37, doi:10.1145/988584.988586.
[92] Microsoft, Kinect for windows 1.5, 1.6, 1.7, 1.8. skeleton class, 2017. https://msdn.microsoft.com/en-us/library/microsoft.kinect.skeleton.aspx.
[93] Logbar, Ring zero, 2017. http://ringzero.logbar.jp/.
[94] O. Bimber, R. Raskar, Spatial Augmented Reality: Merging Real and Virtual Worlds, A. K. Peters, Ltd., Natick, MA, USA, 2005.
[95] Y. Ueda, Y. Asai, R. Enomoto, K. Wang, D. Iwai, K. Sato, Body cyberization by spatial augmented reality for reaching unreachable world, in: Proceedings of the 8th Augmented Human International Conference, in: AH '17, ACM, New York, NY, USA, 2017, pp. 19:1–19:9, doi:10.1145/3041164.3041188.
[96] P. Baudisch, H. Pohl, S. Reinicke, E. Wittmers, P. Lühne, M. Knaust, S. Köhler, P. Schmidt, C. Holz, Imaginary reality gaming: Ball games without a ball, in: Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology, in: UIST '13, ACM, New York, NY, USA, 2013, pp. 405–410, doi:10.1145/2501988.2502012.
[97] S.T. Perrault, E. Lecolinet, Y.P. Bourse, S. Zhao, Y. Guiard, Physical loci: Leveraging spatial, object and semantic memory for command selection, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, in: CHI '15, ACM, New York, NY, USA, 2015, pp. 299–308, doi:10.1145/2702123.2702126.
[98] Vicon, Motion capture systems - vicon, 2018. https://www.vicon.com/.
[99] R. Coppola, M. Morisio, Connected car: technologies, issues, future trends, ACM Comput. Surv. 49 (3) (2016) 46:1–46:36, doi:10.1145/2971482.
[100] M.K. Svangren, M.B. Skov, J. Kjeldskov, The connected car: an empirical study of electric cars as mobile digital devices, in: Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services, in: MobileHCI '17, ACM, New York, NY, USA, 2017, pp. 6:1–6:12, doi:10.1145/3098279.3098535.
[101] P. Di Lena, S. Mirri, C. Prandi, P. Salomoni, G. Delnevo, In-vehicle human machine interface: an approach to enhance eco-driving behaviors, in: Proceedings of the 2017 ACM Workshop on Interacting with Smart Objects, in: SmartObject '17, ACM, New York, NY, USA, 2017, pp. 7–12, doi:10.1145/3038450.3038455.
[102] K.R. May, T.M. Gable, B.N. Walker, Designing an in-vehicle air gesture set using elicitation methods, in: Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, in: AutomotiveUI '17, ACM, New York, NY, USA, 2017, pp. 74–83, doi:10.1145/3122986.3123015.
[103] K. May, T.M. Gable, X. Wu, R.R. Sardesai, B.N. Walker, Choosing the right air gesture: impacts of menu length and air gesture type on driver workload, in: Adjunct Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, in: AutomotiveUI '16 Adjunct, ACM, New York, NY, USA, 2016, pp. 69–74, doi:10.1145/3004323.3004330.
[104] T.M. Gable, K.R. May, B.N. Walker, Applying popular usability heuristics to gesture interaction in the vehicle, in: Adjunct Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, in: AutomotiveUI '14, ACM, New York, NY, USA, 2014, pp. 1–7, doi:10.1145/2667239.2667298.
[105] TwitterInc., Twitter. it's what's happening., 2018. https://twitter.com/?lang=en.
[106] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, NY, USA, 2012.