SAPIENS: Towards Software Architecture to Support Peripheral Interaction in Smart Environments

OVIDIU-ANDREI SCHIPOR, MintViz Lab, MANSiD, University Ştefan cel Mare of Suceava, Romania
RADU-DANIEL VATAVU, MintViz Lab, MANSiD, University Ştefan cel Mare of Suceava, Romania
WENJUN WU, State Key Lab of Software Development Environment, Beihang University, China

We present SAPIENS, a software architecture designed to support engineering of interactive systems featuring peripheral interaction in the context of smart environments. SAPIENS introduces dedicated components for user and device tracking, attention detection, priority management for devices, tasks, and notifications, context-awareness inference, user interruptibility prediction, and device interchangeability that can be instantiated at will according to the needs of the application. To implement these components effectively, SAPIENS employs event-based processing by reusing the core engine of a recently introduced software architecture, EUPHORIA (Schipor *et al.*, 2019), that was specifically designed for engineering interactions in smart environments with heterogeneous I/O devices, and relies entirely on web standards, protocols, and open data-interchange formats, such as `JavaScript`, `WebSockets`, `HTTP`, and `JSON`. This inheritance makes SAPIENS flexible and adaptable to support implementation of diverse application scenarios for peripheral interaction and for a wide variety of smart environments, devices, platforms, data formats, and contexts of use. We present our design criteria for SAPIENS regarding (1) event handling techniques, (2) quality, (3) contextual, and (4) attention-related properties, and describe its components and dataflows that make SAPIENS a specialized software architecture for peripheral interaction scenarios. We also demonstrate SAPIENS with a practical application, inspired and adapted from Bakker's (2013) classical example for peripheral interaction, for which we provide an online simulation tool that researchers and practitioners can readily use to consult actual `JavaScript` code implementing the inner logic of selected components of our architecture as well as to observe live `JSON` messages exchanged by the various components of SAPIENS.

CCS Concepts: • **Human-centered computing** → **Human computer interaction (HCI)**; **Interactive systems and tools**; • **Software and its engineering**;

Additional Key Words and Phrases: Peripheral interaction; Smart environments; Attention; Simulation.

ACM Reference Format:
Ovidiu-Andrei Schipor, Radu-Daniel Vatavu, and Wenjun Wu. 2019. SAPIENS: Towards Software Architecture to Support Peripheral Interaction in Smart Environments. *Proc. ACM Hum.-Comput. Interact.* 3, EICS, Article 11 (June 2019), 24 pages. https://doi.org/10.1145/3331153

1 INTRODUCTION

The vision of smart environments is making inhabitants' lives comfortable by means of smart assisting technology driven by the interoperation of multiple devices that share data and information and collaborate to deliver a pleasant interactive experience to users [17]. To this end, smart

Authors' addresses: Ovidiu-Andrei Schipor, MintViz Lab, MANSiD, University Ştefan cel Mare of Suceava, 13 Universitatii, Suceava, 720229, Romania, schipor@eed.usv.ro; Radu-Daniel Vatavu, MintViz Lab, MANSiD, University Ştefan cel Mare of Suceava, 13 Universitatii, Suceava, 720229, Romania, radu.vatavu@usm.ro; Wenjun Wu, State Key Lab of Software Development Environment, Beihang University, 37 Xueyuan Road, Beijing, 100191, China, wwj@nlsde.buaa.edu.cn.

2573-0142/2019/6-ART11 $15.00
https://doi.org/10.1145/3331153

environments feature remote control of home appliances, predictive and decision-making capabilities, and facilitate communication and interoperation of smart devices through dedicated software architecture and middleware. At the core of such functionalities lies the ultimate goal of delivering users with the information they need, when they need it, and that is presented in the most suitable form that ambient media can take for effective consumption [51], mediated by interactions that feel natural and intuitive [68]. Regarding the latter, interactions based on voice and gesture input have received considerable attention in the Human-Computer Interaction community [30,48,67,68], as these modalities integrate naturally with user behavior and, when properly designed and reflective of actual user preferences [23,45,65,69,77], are performed with little cognitive effort by end-users and, potentially, using little attentional resources. From this perspective, interactions that can be performed at the "periphery of attention" [7–9] are especially relevant for smart environments.

Peripheral interaction can be defined as the totality of perceptions and actions that take place at the periphery of user attention, constituting a fluent part of everyday life routines; see Bakker *et al.* [9] for in-depth discussion. Although these interactions occasionally shift to the center of attention, they quickly revert to their main, original state, which is running silently in the background without requiring conscious cognitive concentration [10]. In fact, one key aspect of peripheral interaction is represented by the tight relationship with personal routines and familiar contexts [7], which make people little aware of the actions occurring at the periphery of their attention, as those actions are driven by unconscious cognitive processes. In this context, perceptions and actions that are performed at the periphery of attention become shortcuts or means rather than conscious goals [7] and, since routines are considered *"the very glue of everyday life"* [62], any attempt to support effective implementation of such routines and action plans through smart technology contributes towards realizing the vision of smart environments [17].

To illustrate the opportunities offered by peripheral input for smart environments, we present the example of Sandra, adopted and adapted from Bakker's [7] introduction of peripheral interaction (see Figure 1 for a visual illustration):

> *"It is Friday evening. Sandra arrives at home and wishes to relax after a hard day at work. She puts her smartphone on the coffee table, sits on the couch, and instructs the smart environment to play her favorite music by using a voice command: "Ambient, play music." The notes of her preferred composer fill the room. After a few minutes, Sandra resumes reading the e-book on her tablet. While she becomes intrigued by the action from the book, she feels that the ambient music is distracting and decides to pause it, "Ambient, stop music," while continuing to read. During this time, various notifications arrive on her smartphone, but because the environment "knows" that Sandra is involved in another activity and that those messages are low priority, they are subtly transferred to a nearby display situated at the periphery of Sandra's attention, instead of the smartphone loudly interrupting her reading. However, when a text message from Sandra's daughter arrives, it is immediately delivered by the surround sound system, while also displayed as a pop-up notification on the tablet, where Sandra is reading her book. Sandra leaves the house to pick up her daughter who has finished her violin lessons a bit earlier today."*

In this example, the environment is aware of Sandra's location, actions, devices, and focus of attention. The surround sound system resides outside Sandra's attentional field at the beginning of the story, but enters the focus of attention when Sandra wishes to relax to her favorite music, fades again at the periphery of attention, and shifts back into focus when Sandra feels distracted by the music and when the important text message from Sandra's daughter is delivered. The tablet also shifts from idle mode to peripheral interaction when Sandra stops the music, and to focused interaction when Sandra reads her e-book or when the important notification is displayed. The

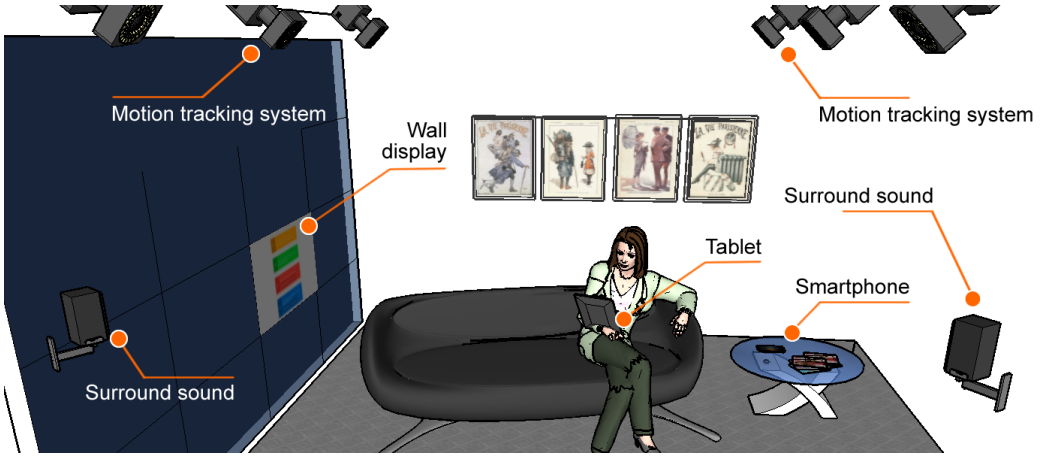

Fig. 1. A visual illustration of a smart environment with multiple input and output devices exemplifying an use case for peripheral interaction adapted from Bakker [7] that we use in this work to introduce, discuss, and demonstrate our Sapiens software architecture. In this figure, Sandra reads an e-book on her tablet, while the smart environment, powered by Sapiens, determines the appropriate output devices to deliver notifications to Sandra based on their priority; see a full description of the scenario in the text and our online simulator from the Sapiens web page.

wall display addresses the periphery of Sandra's attention by showing messages of low priority, made available to Sandra in a way that is not interruptive of her current activity. Note how various devices transition from one state to another depending on (i) Sandra's explicit commands, (ii) events received from third party services, and (iii) decisions taken by the smart environment regarding what content to display and where. Handling these interactions involving heterogeneous devices needs software architecture specifically designed to operate effectively under such conditions. In this work, we address this aspect by proposing Sapiens, a multi-layered, event-based, asynchronous processing Software Architecture for Peripheral Interaction in Smart ENvironments. Our practical contributions are as follows:

(1) We introduce Sapiens, a specialized software architecture to support engineering of peripheral interactions in the context of smart environments. Sapiens integrates recent trends in software architecture design for smart environments, as it builds on core components of Euphoria [58], a software architecture for designing interactions across heterogeneous devices. We present the design requirements of Sapiens, its advancement over Euphoria represented by its super-specialization and focus on peripheral interaction, and discuss the operation logic of its layers, components, and dataflows.

(2) We illustrate the Sapiens software architecture with a practical scenario adapted from Bakker's [7] classical example of peripheral interaction. We provide an online simulated environment for this scenario that enables practitioners to consult actual `JavaScript` code and visualize live `JSON` messages exchanged by the components of the Sapiens architecture.

We introduce Sapiens to foster further research and development in the community of Engineering Interactive Computing Systems towards applications and systems featuring peripheral interaction in smart environments. We hope that Sapiens will generate constructive and fruitful discussion in the interested community, advancing our scientific and practical knowledge in this direction. To this end, we also deliver a companion web page for Sapiens available at the web address http://www.eed.usv.ro/mintviz/resources/SAPIENS.

2 RELATED WORK

We discuss in this section prior work on peripheral interaction and we connect to the wide literature on designing and engineering interactions for smart environments, while pointing to various previous software architectures to highlight the specific contributions of Sapiens.

2.1 Peripheral Interaction

Weiser [75] envisioned the future of information as ubiquitous computing, where all interactions are mediated by devices that fade into the background and become part of the routines of everyday life. On this ground, calm technology [76] was introduced to denote interactive systems capable to address not only the center, but also the periphery of users' attention, while remaining unremarkable [62]. Although very similar to the concept of calm technology, peripheral interaction emphasizes a bidirectional flow of information, from the environment to the user by means of perceptions, and vice versa by means of actions; see Bakker *et al.* [9]. Moreover, the transition between the focus and the periphery of attention can follow different patterns, which in turn have distinct impacts on the attentional flow [70].

Several interactive systems were implemented and evaluated in the scientific literature to examine the opportunities of peripheral interaction [2,3,11,12,67] by introducing prototypes, devices, and applications to deliver users with information at the periphery of their attentional field. For example, the "CowClock" [12] and "FireFlies" [11] interactive devices and use case scenarios were specifically designed to address preschool children. In the CowClock study, children were asked to perform a time-limited task, while the remaining time was marked on a clock using colors and soundscapes providing thus time awareness at the periphery of attention. In the FireFlies study, children had lighting devices shaped as fireflies on their desks to indicate the permission to discuss, give turns, and make compliments. The devices were operated remotely by the teacher to manage the flow of the discussion with children. Both the CowClock and FireFlies studies were conducted during several weeks so that children could become accustomed with those devices. Moreover, the experimental designs relied on informal evaluation methods, such as observations and interviews with children and teachers. These aspects highlight the challenges of evaluating interactive systems designed for the periphery of attention. Another example of peripheral interaction is "Lantern" [2], an interactive lamp designed to mirror the work status of students members of the same team. Several characteristics of light, such as color, intensity, and a blinking effect were used in the lamp design to inform team members about their tasks, elapsed time, and help status from the teacher. The results of the study indicated a positive effect of the lamp devices on students' productivity and communication inside the team. Ambient displays and ambient media [50,51] were also used to deliver information at the periphery of users' attention [3,64,67]. For example, the "Lernanto" system [3] displayed information about the students of a classroom in the form of partially colored octagons; the "Presence Bubbles" visualizations [64] were introduced to display posts from public social networking sites on ambient surfaces at the periphery of attention to enhance human-human communication via ambient media; while the "Audience Silhouettes" [67] represent visual depictions of remote TV viewers' body movements displayed on top of the TV transmission to enhance social television watching by exploiting body kinesics addressing the periphery of attention.

Context modeling represents a key part of peripheral interaction because it offers the conceptual support needed to understand users' activities and focus of attention. Implementation of context-awareness ranges from rules [24,63] to vocabularies [15], and to the design of responsive environments [4]. For example, the "Context Modeling Toolkit" [63] represents a rule-based context processing engine that addresses end-users, expert users, and programmers, while also enabling seamless transition between these user categories; and Chuang *et al.* [15] introduced a vocabulary

composed of 38 items and 12 categories to create language structures with the purpose to mediate communication between users and Internet-of-Things systems.

2.2 Designing Interactions in Smart Environments

The concept of a "smart environment" took shape progressively in the measure that ubiquitous [75], mobile [21], and wearable computing [39] developed as distinct concepts and fields of study. Smart environments consist of sensors, displays, and interactive devices that share the same physical space and collaborate towards achieving the same goal *"to make inhabitants' lives more comfortable"*; see Cook and Das [16] for in-depth discussion. Therefore, several features are expected to be implemented in a smart environment: users express intentions via modalities that are intuitive, natural, and flexible; new devices can be added without cumbersome configuration; and devices share relevant data and information [17]. A similar vision is depicted by the community of Ambient Intelligence (AmI) that is interested in environments *"that pro-actively, but sensibly, support people in their daily lives"*; see Augusto [5]. A subtle, yet important semantic distinction between smart environments and AmI consists in the emphasis that is put on the physical infrastructure in the former case, whereas AmI focuses on implementing the algorithmic behavior and intelligence of the environment [6]. In this work, we use the term "smart environment" to refer to an ecology of interconnected smart devices that impacts users' perception of, attention to, and interpretation of the information, content, and feedback delivered by the smart environment.

Interactions in smart environments can be *explicit*, when users voluntarily initiate communication with the ambient system, or *implicit*, when the environment observes users' activities from which it infers their needs and intents [6,64,66,71]. The ability of a smart environment to detect users and respond appropriately to their actions is provided by the mechanism of *context-awareness* [13,18], where context refers to *"any information that can be used to characterize the situation of an entity"*; see Abowd *et al.* [1]. This ability can be further decomposed into several components, such as *proximity-awareness* encompassing physical parameters [19,33,41,61] or *event-awareness* when referring to various events occurring within the environment [19,26,42]. Proximity-awareness is particularly relevant to Sapiens because the relative position of users with respect to given objects has often been associated with attention to those objects [28,41,61]. Several measures can be employed to operationalize the concept of proximity, such as the distance between users and objects [28,61], orientation measurements [28,74], or movement, such as pointing to objects from the environment [52,57,68]. These examples show that there is a need for a specific *attention-awareness* component as part of the context-awareness capability of a smart environment to manage inferences about users' focus and periphery of attention. With such a component, a dedicated software architecture could deal effectively with application scenarios involving peripheral attention and interaction. Moreover, as the data collected by the environment refers to diverse aspects, such as proximity, health, user behavior, or emotional and cognitive states [44,55], the smart environment needs to run complex algorithms to infer the needs and intentions of its inhabitants [35]. For instance, emotion recognition may require multimodal analysis, fusion, and classification [20,54,56].

Besides proxemic measurements, other techniques have been proposed to detect users' focus of attention. For example, systems based on eye tracking can detect visual attention due to the relationship between eye gaze and the entities within the environment [40,72,73]. "WISEglass" is a platform that relies on smartglasses to understand context from a user-centered perspective [73]. The analysis of contextual elements, such as position, speed, and saliency of objects within the user's visual field, is used to estimate the focus of attention [37]. The WISEglass approach was evaluated for activity recognition and screen-use detection with accuracies of 77% and 80%, respectively. The "SwitchBack" [40] technique uses the front-facing camera of a mobile device to track the user's

eye gaze on the screen. When the user looks away from the device and then back, SwitchBack highlights the most recently read body of text so that attention can be easily restored.

2.3 Software Architecture Design and Engineering for Smart Environments

We continue with a discussion of previous software architectures and systems implementing interactions in smart environments. The most relevant prior work is Euphoria [58], a scalable, event-driven software architecture designed to handle interactions with heterogeneous devices in smart environments. Euphoria has five layers: Producers, Emitters, Engine, Receivers, and Consumers. Producers and Consumers represent all the devices that operate in the smart environment. Emitters and Receivers are software interfaces between Euphoria's Engine and devices implementing Producers and Consumers. The Engine processes and dispatches messages shared by the previous components. We discuss Euphoria in detail in the next section, where we present our design criteria for the Sapiens software architecture and show how Sapiens advances over Euphoria with specialized components for peripheral interaction, while building on Euphoria's efficient engine for producing, exchanging, and consuming messages.

The "Proximity Toolkit" [28,41] is a proxemic-aware tool that operationalized the concept of proxemic interaction using specific measures of location, orientation, distance, motion, and identity of users and devices from a smart environment. The authors resorted to several application scenarios to illustrate the functionality and flexibility of the Proximity Toolkit. For example, the content displayed on the TV can automatically change size and position on the screen in response to changes in users' location and orientation in front of the TV screen; also, a system implementing proxemic interaction could detect whenever users are engaged in a conversation and use this information to pause media. In these scenarios, the authors implicitly assume a relationship between proxemic measures and the users' focus of attention, which is a reasonable assumption, also adopted in our implementation of the Sandra use case scenario for Sapiens; see Section 4. The Proximity Toolkit comes with an API (Application Programming Interface) to enable rapid prototyping of new proxemic-aware interactive systems. Its architecture is built around a server that fuses and processes proximity measurements, while the generic data model is decoupled from the hardware implementation so that the toolkit can be easily adapted to work with diverse sensing technology for tracking users and objects in the smart environment. An earlier version of proxemic interaction was proposed by Tănase *et al.* [61] for interactive surfaces. In their demonstrative application, the content displayed by an interactive tabletop changes its orientation automatically to follow users moving around the table. The "Proxemic Peddler" [74] is another example of a prototype that monitors the passerby's distance and orientation with respect to an ambient display. The system infers the passerby's attention based on their short-term interaction history with the display and then uses this information to adjust the content in order to encourage further engagement with the ambient display.

Other software architectures and systems were proposed to support interactions in smart environments. "XDKinect" [48] and "Mobile@Old" [44] are two examples that use the Kinect sensor [43], a device not supported any longer by Microsoft, but invaluable for rapid prototyping and evaluation of whole-body gesture interaction. Both systems feature modular, decoupled, event-driven, and smart space-oriented designs. "BodyCloud" [22] is a Software as a Service (SaaS) approach for networks of body sensors. A BodyCloud application supports three types of clients: the Body (*i.e.*, a component that collects data from wearable sensors), the Viewer (*i.e.*, a module that implements graphical output), and the Analyst (represented by components that support development of new BodyCloud applications). The main advantages of the BodyCloud architecture are its scalability, interoperability, and support for the development of new applications. The "SoD-Toolkit" [59] implements complex spatial-awareness scenarios involving multiple sensors and multi-device

environments. The toolkit exposes a `JavaScript` API to implement communications with native applications and 3-D localization. The "Gator Tech Smart House" was introduced by Helal *et al.* [31] to extend the perspective on pervasive spaces towards including not only a runtime environment, but a software library as well. According to this perspective, third-party applications could extend and configure the original system by adding extra components and upgrading its software services.

An important aspect of software architectures for smart environments regards the type of content and messages that can be exchanged between devices. For example, the same notification can be delivered via different channels, *e.g.*, visual, aural, haptic, and by different devices. To this end, multimodal rendering techniques [53] could be used to infer dynamically the most suitable device, modality, and format to present notifications to users. Then, it is the users' responsibility to change between devices and input modalities as they deem fit [14]. Ideally, the users of a smart environment should perceive the environment as a single, holistic entity [14].

3 SAPIENS

We introduce in this section Sapiens, our proposal for software architecture to support engineering of peripheral interactions [7,9] for application scenarios and use cases pertaining to smart environments, such as the Sandra scenario discussed in Section 1. Sapiens builds on top of the processing engine of Euphoria [58], a generic, event-based software architecture for implementing interactions across heterogeneous I/O devices. Due to this inheritance, we start this section by overviewing the core aspects and functionality of Euphoria, and continue by enumerating and discussing design principles and requirements that are specific to Sapiens as well as new, key software components and dataflows that make Sapiens a super-specialized architecture with an explicit focus on peripheral interaction.

3.1 Overview of the Euphoria software architecture

Euphoria [58] is a recently released software architecture design that leverages open technologies, such as Internet protocols and standards, to enable easy prototyping, deployment, and performance evaluation of user interactions across heterogeneous I/O devices in smart environments. Its design relies on five layers, which include Producers, Emitters, Receivers, Consumers, and the Core Engine that handles the exchange of messages. Producers and Consumers represent the totality of I/O devices that operate in the smart environment*e.g.*, the smartphone, tablet, microphone, wall display, motion tracker, and surround sound system from the Sandra example, while Emitters and Receivers are software modules that interface Euphoria's Engine, Producers, and Consumers, enabling these components to operate and cooperate effectively. The Engine is implemented by a `Node.js` server[1] that processes and dispatches `JSON`-encoded messages between various entities that either create and transmit messages or are subscribed to specific events. Euphoria implements the following ten design criteria (see Schipor *et al.* [58] for details), which we briefly summarize below to highlight their relevance to peripheral interaction by relating to our Sandra's example from Section 1:

(1) Two **handling techniques** (referred to as H_1 and H_2) implement the processing flow of events in the Euphoria software architecture:
   H_1. *Event-driven processing.* Event processing is triggered by physical, hardware, or software changes that occur in the smart environment, *e.g.*, a swipe gesture performed by Sandra on her tablet to advance to the next page of the e-book, or a text message received on Sandra's smartphone determine further actions from other components of the smart environment that have previously subscribed to receive those events.

[1]Node.js, https://nodejs.org/en/

H_2. *Asynchronicity*. Events are processed in a way that is adapted to the unpredictable time-response behavior of the entities from the smart environment that, most likely, need to share data via Wi-Fi networks, which are affected by congestion and latency issues [47]. Moreover, it is quite an engineering challenge to synchronize the clocks of several devices [38], such as the smartphone, tablet, and the wall display from our example, that may run different operating systems and embed electronic components that synchronize using different clock circuits. Taking actions based on events that are received and processed asynchronously make applications little affected by differences in time synchronization between devices.

(2) Four **quality properties** (Q_1 to Q_4) control how the various components of the Euphoria architecture (*i.e.*, Producers, Consumers, Emitters, Receivers, and the core Engine) relate to each other, as follows:

Q_1. *Adaptability* (or *scalability*). Euphoria can handle structural changes, such as the inclusion of new components that are added to the architecture (*e.g.*, components that implement operations on data streams or formats specific to the input devices considered by a specific application) or removal of components that are no longer needed (*e.g.*, components for retrieving data from input devices that become obsolete in time). This property is useful for application scenarios involving peripheral interaction, where new I/O devices are added and removed automatically to/from the processing dataflows, *e.g.*, while Sandra moves from one room to another, the list of devices located in the vicinity of Sandra that are available to deliver notifications at the periphery of Sandra's attention is constantly updated.

Q_2. *Modularity*. The decoupling of Euphoria prevents that changes occurring in one module to propagate throughout the entire architecture. As we will show later in this section with a practical `JavaScript` code example, the tablet device that Sandra employs to read her e-book can be modeled and implemented effectively using Producer and Consumer components from Euphoria and further changes to the tablet implementation will not affect the processing dataflows from other parts of the architecture.

Q_3. *Flexibility*. Euphoria can be used reliably, with minimum software changes, in scenarios that exceed the original specifications. For example, the developers of Sandra's smart home environment may wish to provide a new functionality, *e.g.*, adding a pair of smartglasses to the scenario to augment the original functionality of the wall display. In that case, the new smartglasses object can be readily implemented as a new Consumer using the formalism of Euphoria's components, while the wall display object rests unchanged in the list of active devices from Sandra's physical environment.

Q_4. *Interoperability*. Euphoria can readily exchange data with external systems when integrated in larger software architectures. The ecology of devices from Sandra's scenario includes a tablet, a smartphone, a wall display, a surround sound system, a microphone, and a motion tracker. While some of these devices collect input from the physical environment, *e.g.*, the microphone and the motion tracker, other devices need to connect to external, third-party services to receive data and notifications, such as Sandra's smartphone. Using Euphoria, the source of an event is not important as long as events are being encoded and transmitted according to a formalism that specifies correctly the content of the message and its origin. For example, a voice command issued by Sandra is picked up by the microphone from the physical environment, but processed using an online speech-to-text service before being delivered to the Engine.

(3) Four **contextual properties** (C_1 to C_4):

C_1. *Web-based standards and protocols.* Euphoria employs common Internet standards and protocols for data transmission, such as `JSON` and `WebSockets`. This is an important aspect to facilitate easy accommodation of different devices, *e.g.*, smartphone and wall display, that, despite many differences, implement the same set of Internet-based standards and protocols for sharing data.

C_2. *Open-source technology.* Euphoria commits to open technology to foster development and innovations regarding user interactions in smart environments. This principle is equally valid for Sapiens, an architecture that we deliver to the community to foster more research and development regarding peripheral interactions.

C_3. *Smart spaces orientation.* Euphoria can handle complex interactions between users, devices, and physical objects in a smart environment, a property that we equally envisage for the Sapiens architecture.

C_4. *JavaScript-based.* The default language employed by all the components of Euphoria is `JavaScript`, although any other programming language that supports web technologies can be equally employed. According to Stack Overflow's Developer Survey Results from 2018 [60], `JavaScript` is the most commonly used programming language, both among professional developers and all respondents, for the sixth year in a row. Therefore, we align Sapiens to this trend and growing popularity of `JavaScript` to foster access to, adoption of, and reutilization of our architecture as much as possible.

3.2 From Euphoria to Sapiens

As exemplified in the previous section, the design criteria and requirements of Euphoria are also relevant when engineering systems to support peripheral interactions in smart environments, often involving a diversity of I/O devices. This observation makes Euphoria our software architecture of choice on which to build Sapiens. For example, data regarding the user's focus of attention are available each time when changes occur in the physical state of the user, as detectable by some sensor, *e.g.*, the user's head or arms point to a specific region in space (detectable by a motion tracking device, either worn or installed in the environment); eyes look away, drawn by some new information (detectable by an eye tracker device); a swipe is articulated on the touchscreen of the tablet device; or the auditory cortex of the brain becomes suddenly more active (revealed by electroencephalography measurements delivered by a headset EEG device). To detect such events, a diversity of sensors and input devices are needed, which differ in terms of platforms, protocols, data formats, sampling frequency, software development kits and programming languages to access data, and so on. Since Euphoria already facilitates easy integration of heterogeneous I/O devices, building on top of such relevant previous work represents a solid foundation for our approach to create new, specialized software architecture to support engineering of peripheral interaction in smart environments.

Other features of Euphoria recommend it for our goals, as follows. For instance, the information regarding the user's focus of attention needs to be processed in an event-driven and asynchronous manner: the *event-driven* approach (H_1) enables real-time handling of events that occur in the smart environment, while *asynchronous* processing (H_2) is particularly adapted to unknown time duration of those events. Moreover, a smart environment is usually fluid in terms of the entities that produce and consume events and, consequently, users, devices, and physical objects should be allowed to enter and leave the environment without going through manual registration processes. Correspondingly, the attention-aware module needs to be *scalable* (Q_1) to readily integrate new input devices. This desideratum can be achieved by specifying a comprehensive set of standardized software interfaces so that data delivered by new devices can be incorporated in the attention

inference process. Such interfaces play a key role in the *modularization* of the Sapiens architecture (Q_2) due to the decoupling of the corresponding software components. Just like Euphoria, Sapiens should be *flexible* (Q_3) to implement a variety of application scenarios that exploit users' attentional field, such as new applications or contexts of use beyond the original requirements. Also, it is desirable for an attention sensing environment to *interoperate* (Q_4) with other systems and, therefore, to integrate more complex architectures. For that reason, *web standards and protocols* (C_1), such as the `JSON` format and the `JavaScript` language (C_4), represent preferred choices for Sapiens as well. Moreover, relying on *open standards and technologies* (C_2) can encourage researchers and practitioners to use our work for development of applications that feature peripheral interaction. Since the technical evaluation and discussion conducted by Schipor *et al.* [58] already showed that Euphoria had complied to the above criteria more than any other architecture or system from the literature [22,27,28,41,44,48], we select Euphoria as our event-based software architecture of choice for creating, encoding, and exchanging messages, on which to further build the specialized software components of Sapiens for engineering peripheral interactions.

Before moving further to the details of the Sapiens architecture, we briefly illustrate with a practical example how Euphoria can be used to implement a component that creates, transmits, and consumes messages. Figure 2 illustrates a `JavaScript` implementation of the tablet device from Sandra's example. The `Tablet` is both a Producer and a Consumer: it produces touch input events, letting other components of Sapiens know when Sandra is using her tablet, and it consumes notification messages that need to be displayed, such as the text message received from Sandra's daughter. The `JavaScript` code implements two `EuphoriaConsumer` and `EuphoriaProducer` objects that handle the reception and transmission of messages via the dedicated `onMessageRead(..)` and `onMessageWrite(..)` functions. The code displayed in Figure 2 is an excerpt from our simulator web page and, therefore, generates touch points with random coordinates.

3.3 Design Principles for the Sapiens software architecture

So far, we showed that borrowing design requirements, software components, and dataflows from a prior architecture is beneficial for Sapiens. In the following, we continue our discussion by listing specific design criteria for Sapiens, which enables us to sketch the blueprint of the layers, components, and dataflows of the Sapiens software architecture in the next subsection.

One central requirement of any interactive system implementing peripheral interaction is to infer the user's focus of attention. To this end, a smart environment needs to collect various data from heterogeneous input devices, as exemplified before, process that data, make inferences, and come up with an estimate, most likely expressed in a probabilistic form, regarding (1) the *object* onto which the user is focusing their attention, (2) the corresponding *sensory channels* involved during this process, and (3) the *cognitive load* that the user allocates to the object of attention. Computation models of attention that can deliver satisfactory responses regarding the process of attention are challenging to implement [25,32] and Sapiens does not aim to solve such hard problems from the literature. Instead, Sapiens offers a general framework in which various possible technical solutions can be integrated and evaluated in the context set out by a specific application and/or context of use, such as the Sandra example discussed in Section 1. For example, when Sandra is reading the e-book on her tablet, the system can infer from the tablet touchscreen as well as from the relative orientation of Sandra and the tablet that the primary visual attention is devoted to the e-book, to which Sandra allocates cognitive processing.

Besides the handling techniques (H_1, H_2), quality (Q_1 to Q_4) and contextual properties (C_1 to C_4) borrowed from Euphoria, we propose the following four **attention-related properties** (A_1 to A_4) for the Sapiens architecture that are specific to peripheral interaction and not implemented by default under Euphoria:

```
class Tablet {
  // creates a Tablet device acting both as Producer and Consumer of messages
  constructor(outputURL, inputURL, deviceName, deviceIP, events) {
    this.consumer = new EuphoriaConsumer(
      outputURL, events, this.onMessageRead.bind(this)
    );
    this.producer = new EuphoriaProducer(
      inputURL, "PersonalTouchscreen", deviceName, deviceIP, this.onMessageWrite
    );
    this.message = this.producer.message;
    this.onMessageWrite = this.onMessageWrite.bind(this);
  }

  // Called when sending a new message
  // For simulation purposes, we generate touch points with random coordinates
  onMessageWrite() {
    this.message.header.eventName = "onTouchDown";
    this.message.body.location = {};
    this.message.body.location.x = getRndInteger(0, 480);
    this.message.body.location.y = getRndInteger(0, 640);
  }

  // Called when subscribed events are available
  onMessageRead(message) {
    if (message.header.eventName === "onNotification")
      this.showNotification(
        message.body.modality, message.body.content, message.body.priority
    );
  }

  showNotification(modality, content, priority) {
    // application specific implementation, not shown here
  }

  start() { this.producer.start(); }
  stop() { this.producer.stop(); }
  pause() { this.producer.pause(); }
  resume() { this.producer.resume(); }
}
```

Fig. 2. Code snippet (`JavaScript`) implementing a tablet device in Sapiens using the Producer and Consumer components of Euphoria. With Sapiens, we add on the building blocks of Euphoria to create super-specialized software components to support engineering of peripheral interaction; Section 4 resumes the illustration of code snippets to demonstrate an implementation of the Sandra's use case scenario using Sapiens. *Note:* touch events are simulated at random locations on the touchscreen; see the online simulator for this code.

- A_1. *Multimodal orientation.* Processing of events and decision making in Sapiens should reflect the multimodal nature of human attention and, consequently, offer an appropriate conceptual framework for peripheral interaction to address not just one sense, but multiple senses at once towards a rich user experience. For example, when an important, high-priority notification arrives on Sandra's smartphone, that message will be delivered using visual and aural feedback via the smartphone, surround sound system, and the tablet on which Sandra is reading her e-book. Other, low-priority notifications are delivered in ways that do not interrupt Sandra's current reading, such as a gentle rendering of the notifications on the wall display accessible to Sandra's periphery of attention.
- A_2. *Priority inference* represents the ability of Sapiens to distinguish various demands for the user's attention coming from various devices and prioritize what information to deliver users.

For example, low-priority notifications should not disturb Sandra's reading, while important messages must address Sandra's focus of attention using devices that already capture her attention or are in the immediate vicinity of Sandra's attentional field, such as the tablet and the surround sound system in our example.

A_3. *Probabilistic response.* Human attention can be distributed across several tasks, each having a specific cognitive load. For example, when Sandra is reading on the tablet and listening to the music delivered via the surround sound system, her attention is obviously distributed. However, it is reasonable to assume that the reading activity demands more cognitive resources and, therefore, is the primary task that captures Sandra's focus of attention. Therefore, a probabilistic characterization of the attentional process is recommendable, for which we can propose a simple mathematical formalism. Let $P(d_i)$ denote the probability that device d_i from the environment has captured Sandra's focus of attention under the constraint of normalized probabilities, $\sum_{d_i} P(d_i) = 1$. If a touch event comes from the tablet, then $P(tablet)$ increases, as the environment gets confidence that Sandra is operating the tablet. If a voice command to turn music on is issued by Sandra, then $P(surround\text{-}sound\text{-}system)$ increases on the premises that Sandra will pay attention to the music that is about to play. The specific handling of such probabilities depends on the actual devices, types of sensors and data that can be collected to make inferences regarding the user's attention and, ultimately, the requirements of the application. Later in the article, we show how these probabilities can be reasonably estimated for the Sandra's example by starting from a default value, which increases each time the system builds up confidence regarding the device likely to capture Sandra's focus of attention; see Figure 7 for a `JavaScript` implementation of this technique. The probabilistic response design is a new, specialized component, not available in Euphoria, which was not meant to rank devices based on probabilistic formalisms.

A_4. *Proactivity* underlines the capacity of Sapiens to infer not just the user's focus of attention, but also to predict changes. This design requirement is important for an application to know when to switch between the main and secondary tasks. Our suggestion to implement this requirement is through the formalism of "hazard functions," which describe the probability of an event, in our case Sandra's attention to the tablet device, to terminate in the immediate future. We inspire from Hawkins *et al.*'s [29] empirical observations regarding a hazard function for modeling looks at television, which peaks at about 1-1.5 seconds and then decreases towards a very low asymptote after about 15 seconds; *e.g.*, looks at a device, such as the TV, that has attracted user's visual attention, tend to gain inertia in the form of likelihood to continue as time elapses. Put formally, the proactivity requirement is implemented by a hazard function $\lambda(t) : (0, \infty) \rightarrow R^+$ that integrates to unity, $\int_0^\infty \lambda(t) = 1$. While the implementation is specific to the device and the application, Section 4 demonstrates a technique using a hazard function decreasing in geometric progression; see Figure 7.

3.4 Description of the Sapiens Architecture

Figure 3 presents a visual illustration of the layers and components of the Sapiens software architecture, together with specific instances of I/O devices representative of the Producers and Consumers of a smart environment implementing peripheral interaction. The figure also shows the role played by Euphoria's Engine, which is employed by Sapiens to exchange messages effectively between its various components. Sapiens consists of the following components:

(1) Devices represent specific instances of input and output devices, which are referred to as Producers and Consumers in the original Euphoria architecture [58], a terminology that we borrow and employ for Sapiens as well. Input devices capture relevant data from the

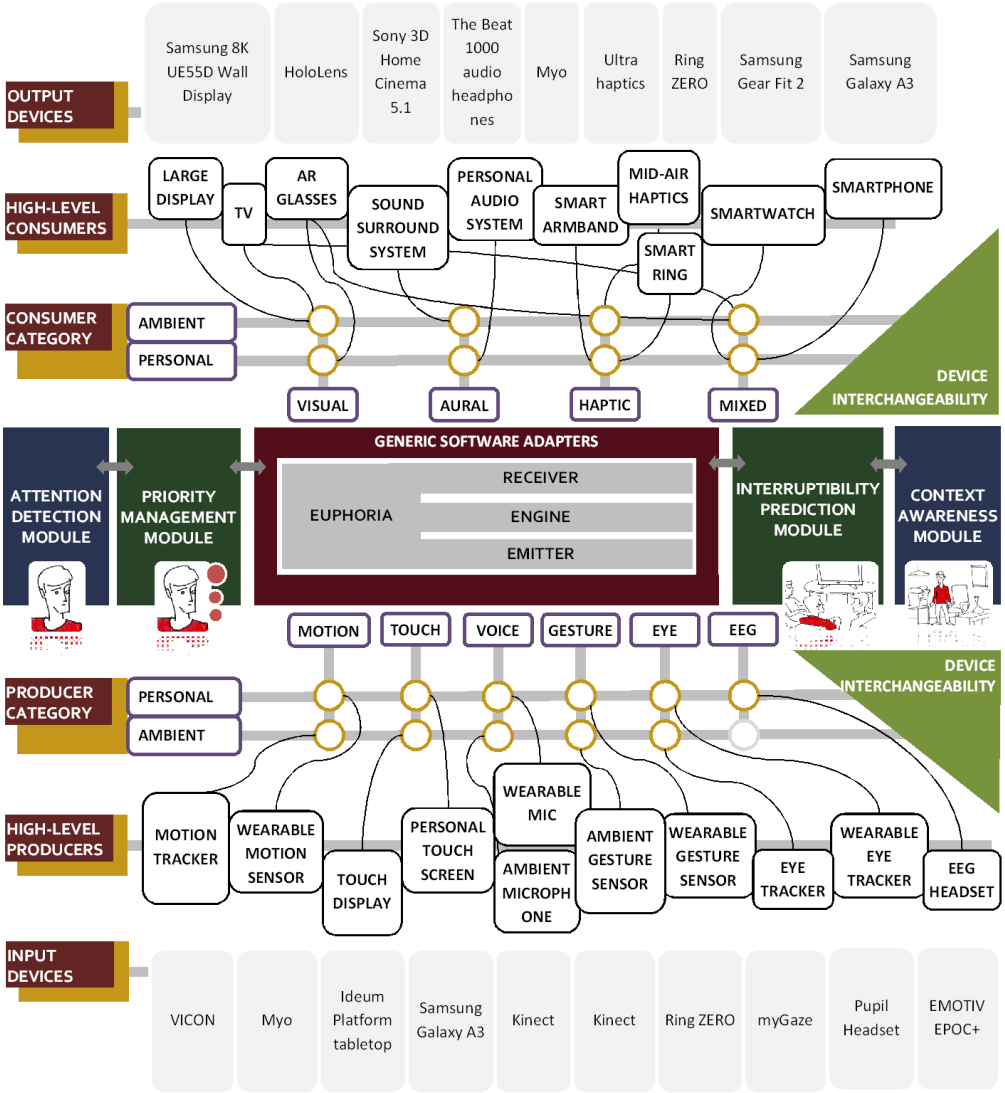

Fig. 3. Visual illustration of the SAPIENS multi-layer software architecture.

smart environment and expose events, while output devices present users with feedback, based on the content of the messages delivered with the events. For instance, the *Sony 3D Home Cinema* and the *Beat 1000 audio headphones* are two examples of output devices that can deliver aural feedback when Sandra's listens to her favorite music or when an important, high-priority notification needs to be delivered; see the description of our example scenario from Section 1. *Vicon* and *myGaze* are examples of Producers because they collect measurements and transmit data regarding the users of the smart environment: *Vicon* [34] detects the location, motion, and orientation of any entity (*e.g.*, Sandra, her smartphone and tablet, etc.) by monitoring infrared markers attached to those entities, and *myGaze* [46] is a personal device that performs binocular gaze tracking reporting information regarding the visual focus of attention. Other devices act as both Producers and Consumers or target

multiple sensory modalities at once. For example, the *Samsung Galaxy A3* smartphone detects touch input and shares it to other components of Sapiens via messages and events and delivers visual, aural, and haptic feedback in response to messages corresponding to events to which the tablet device has previously subscribed. Likewise, smart armbands and smart rings act as both input and output devices by collecting arm motion and gestures and by providing vibrotactile feedback.

(2) High-Level Devices group similar devices together to enable implementation of generic data formats and a common business logic. They are especially important for attaining the *modularity* and *flexibility* desiderata for Sapiens (quality properties Q_2 and Q_3), because they maintain the software architecture free from frequent changes. The goal is to reduce, as much as possible, the practitioner's workload to add a new device and, once part of a High-Level Device group, the implementation of a new device can readily inherit the existing functionality of a High-Level Device component. In this context, using a new *smartphone* should imply nothing more than the creation of a `JSON` file to describe the availability of its embedded sensors (*e.g.*, accelerometer, GPS, microphone) and the type of feedback that can be delivered using that smartphone (*e.g.*, visual, aural, vibrotactile). Because some devices feature both input and output capabilities, they can be part of multiple High-Level Consumer and High-Level Producer groups. The *Samsung Galaxy A3* device, for instance, belongs at the same time to both the *Smartphone* and *Personal Touchscreen* groups. This is also true for the *Myo armband* [49] (with corresponding groups *Wearable Motion Sensor* and *Smart Armband* due to its motion and gesture sensing features, but also its form factor designed to be worn on the user's forearm) and for the Ring ZERO device [36] (*i.e.*, the *Wearable Gesture Sensor* and *Smart Ring* groups).

(3) Device Categories connect High-Level Devices with specific consumer category states, which can be either *personal* or *ambient*, and modalities (*visual*, *aural*, *haptic*, and *mixed*); see Figure 3. Consumers are linked to modalities, while Producers to human actions, *e.g.*, *motion*, *gestures*, *voice*, *EEG*, etc. We also introduce the *mixed* category to address devices that target multiple sensory channels, *e.g.*, *smartphones* can deliver messages using visual, aural, and vibrotactile stimuli. However, not all the categories resulting from the intersection of consumer and modality types actually contain devices, such as the case of *ambient* and *EEG*, for which no devices currently exist, to the best of our knowledge. Please note that the categories presented in Figure 3 are not exhaustive and new device categories could be further added to the architecture.

(4) The Generic-Software-Adapters module keeps Sapiens decoupled so that changes in one component do not propagate throughout the entire architecture. The role of the adapters is to assure the *interoperability* with other systems (quality property Q_4). Moreover, they enable generic implementation of the attention-related software components, such as the Attention-Detection and the Priority-Management modules. Generic adapters also convert different data formats to `JSON`, which is the native representation for messages that are created, transmitted, and processed by Euphoria and, implicitly, by Sapiens.

(5) Euphoria represents the core of the Sapiens software architecture and is used by Sapiens to deliver events and messages effectively to the various subscribing components. The Engine receives events from Producers (input devices), dispatches them to the attention-related modules, and forwards the results to Consumers (output devices). Euphoria revolves around efficient production, transmission, detection, and consumption of device- and environment-specific events, which are features that make it the foundation on which Sapiens is implemented. Schipor *et al.* [58] presented experimental results regarding the request-response

time performance of the Euphoria architecture under various conditions of message size, environment complexity, and device capabilities.

(6) The Attention-Detection-Module represents the component that estimates the user's focus of attention in the smart environment. This component receives events from input devices and infers an association between the user's focus of attention and a specific device. In our example, this module can infer the primary task by analyzing messages from the Vicon motion tracking system to match Sandra's head orientation with the location and orientation of the tablet and gain confidence in the inference that the tablet is the focus of attention.

(7) The Priority-Management-Module receives messages from the Attention-Detection-Module via the Engine and implements the business logic of the application running in the smart environment. The priority of a device is dependent on the specific scenario and the content to be delivered to the user. Resuming our example, consider two devices, the tablet and the wall display competing for Sandra's visual attention. The Priority-Management-Module is the only component responsible for establishing priorities based on the content displayed on each device. For example, the wall display may have a lower priority than the tablet, as a default setting. However, if an important message needs to be delivered, *e.g.*, the text message from Sandra's daughter, the Priority-Management-Module may use both devices to attract Sandra's attention.

(8) The Interruptibility-Prediction-Module handles all the external events that compete for the user's attention. It interrogates the Priority-Management-Module, via the Euphoria Engine, to receive information about the user's focus of attention and the priority of each device. This module establishes the suitable device(s) and modality (or modalities) for notifying the user effectively. In the case of a new text message, the Interruptibility-Prediction-Module determines what device has the user's primary attention, and decides whether the event needs to be delivered on that device by comparing the relative priorities of the current task (*i.e.*, Sandra reading on the tablet) and the new task (a text message from Sandra's daughter). Since the text message is more important, according to the default setting, the Interruptibility-Prediction-Module creates a new event and message that will travel through the architecture to the tablet device.

(9) The Context-Awareness-Module collects contextual information from the environment, such as the location and orientation of entities, which it shares with the Interruptibility-Prediction-Module via the Engine. This way, it is possible to deliver an important message not only through the device that has already captured Sandra's attention, but also using other Consumers that are conveniently located and oriented in the vicinity of Sandra. This module communicates with the Attention-Detection-Module via the engine.

(10) Device-Interchangeability addresses multiple modalities and devices through which a message can be delivered to users. This component is closely related to the Interruptibility-Prediction-Module as it allows the adaptation of each message to each device and to the user. In our example, the same notification can be delivered to Sandra via the tablet, but also by reading and delivering it via the surround sound system.

4 RESUMING THE SANDRA EXAMPLE: AN ONLINE SIMULATOR FOR SAPIENS

We resume in the following our example from Section 1 regarding Sandra's interactions with the smart environment. Figure 4 illustrates a block diagram with selected components from Sapiens that are needed for this scenario: input devices (acting as Producers), output devices (acting as Consumers), and the Attention-Detection, Priority-Management, Context-Awareness, and Interruptibility-Prediction modules that exchange messages via the Euphoria Engine. Our

goal in this section is to demonstrate how the various components of the SAPIENS architecture can be instantiated for the implementation of the Sandra scenario. To this end, we introduce our online simulation application for this scenario written entirely in HTML5, JavaScript, and the three.js library,[2] which enables practitioners to have access to actual running code and to observe live JSON messages that are exchanged by the various components of SAPIENS; see Figure 6 for a screenshot of the simulator.

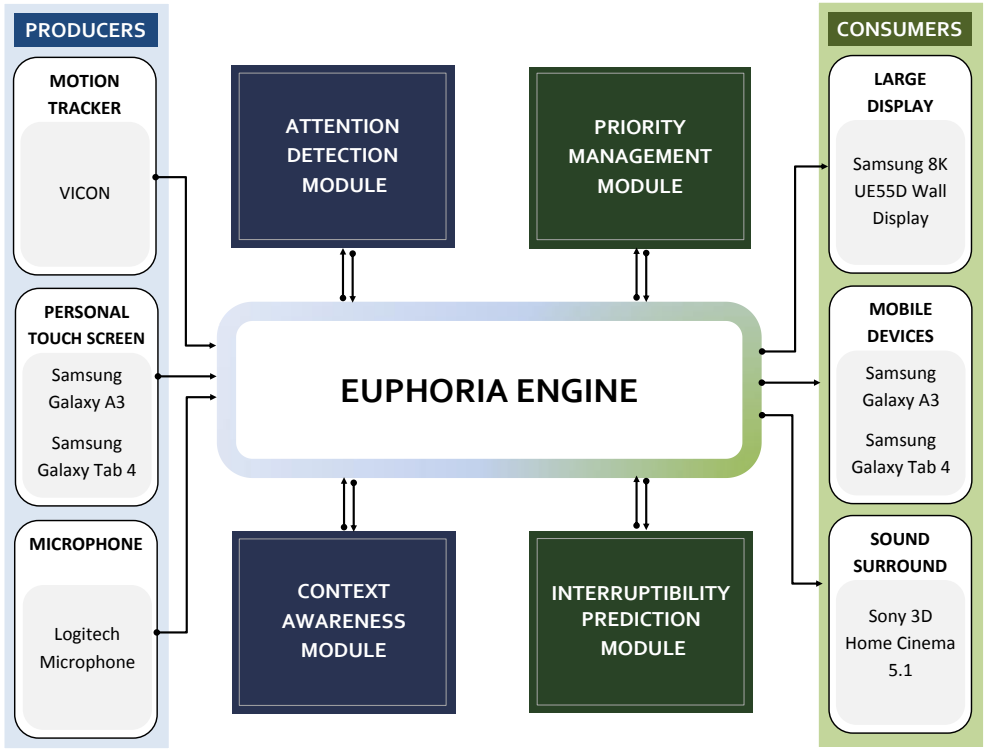

Fig. 4. Devices, components, and dataflows from SAPIENS selected to implement the Sandra scenario; see the full description of this scenario in Section 1.

4.1 Devices, Producers, and Consumers

The Sandra scenario introduces several I/O devices: the motion tracking system, smartphone, tablet, wall display, environmental microphone, and surround sound system. In our simulator, we implemented each of these devices as Producers, Consumers, or both, depending on their assigned functionality. For example, the Vicon motion tracker is implemented as a Producer, while the Tablet acts both as a Producer that signals touch input events to the other components of the architecture and as a Consumer when it displays the important text message notification received by the Smartphone. Figure 2 shows an implementation of the Tablet object in our simulator. For simulation purposes, touch events are randomly generated on the tablet screen at a specified time interval, but the communication with the Engine is not affected by the fact that data are simulated. The other devices are implemented in a similar manner: Vicon, Smartphone, Microphone,

[2]Three.js is a lightweight, cross-browser JavaScript library for displaying animated 3-D computer graphics in a web browser using the default WebGL renderer; see https://threejs.org and https://github.com/mrdoob/three.js.

```
{
  "header": {
    "deviceName": "Vicon",
    "highLevelDevice": "MotionTracker",
    "deviceIP": "192.168.0.7",
    "eventName": "onMotionTrackingData",
    "timestamp": 1555916839619
  },
  "body": {
    "entities": [
      {
        "name": "Sandra",
        "location": {"x": 915, "y": 2707, "z": 1470},
        "orientation": {"yaw": 0.437, "pitch": -0.538, "roll": -0.600}
      },
      {
        "name": "WallDisplay",
        "location": {"x": 618, "y": 1696, "z": 1247},
        "orientation": {"yaw": -0.767, "pitch": -0.254, "roll": -0.412}
      },
      {
        "name": "Smartphone",
        "location": {"x": 219, "y": 459, "z": 962},
        "orientation": {"yaw": 0.660, "pitch": 0.336, "roll": 0.229}
      }
    ]
  }
}
```


Fig. 5. Example of (an excerpt of) a `JSON` message produced by the `Vicon` object. In this example, simulated tracking data (location and orientation) are shown for three entities from the smart environment: Sandra, `WallDisplay`, and `Smartphone`.

`WallDisplay`, and `SurroundSound` represent `JavaScript` classes and instantiated objects in our online application. Figure 4 lists names of actual devices that can be used to implement the functionality of our simulated objects, *e.g.*, the "Sony 3D Home Cinema 5.1" for the `SurroundSound` system or the "Samsung Galaxy AR3" for the `Smartphone`. Figure 5 illustrates an example of a `JSON` message produced by the `Vicon` object that respects the format imposed by the Euphoria architecture and inherited by Sapiens, *i.e.*, a *header* with identification data regarding the device and the event and a *body* with the actual data; see Schipor *et al.* [58] for more details and examples and our online simulator for the complete set of `JSON` messages for the Sandra use case scenario.

4.2 Dataflows

The four specialized components of Sapiens (Attention-Detection, Priority-Management, Context-Awareness, and Interruptibility-Prediction) receive messages from the Engine according to the events they have previously subscribed to and produce messages in return; see the dataflows illustrated in Figure 4. Two devices act as Producers (`Vicon` and `Microphone`) and deliver messages to the Engine; two devices act as Consumers (`WallDisplay` and `SurroundSound`); and two devices are both Producers and Consumers (`Smartphone` and `Tablet`). For example, the data collected from `Vicon` (in the form of location and orientation of the entities registered in the smart environment) and `Tablet` (touch input events) are delivered by means of the Engine to the Attention-Detection-Module, which produces a probabilistic estimate of the device most likely to capture Sandra's focus of attention; see Figure 7 for actual `JavaScript` code and our discussion from the next section. Default priorities for each device are available to the simulator according to the specifics of Sandra's scenario, *i.e.*, `Tablet` is high priority, followed by

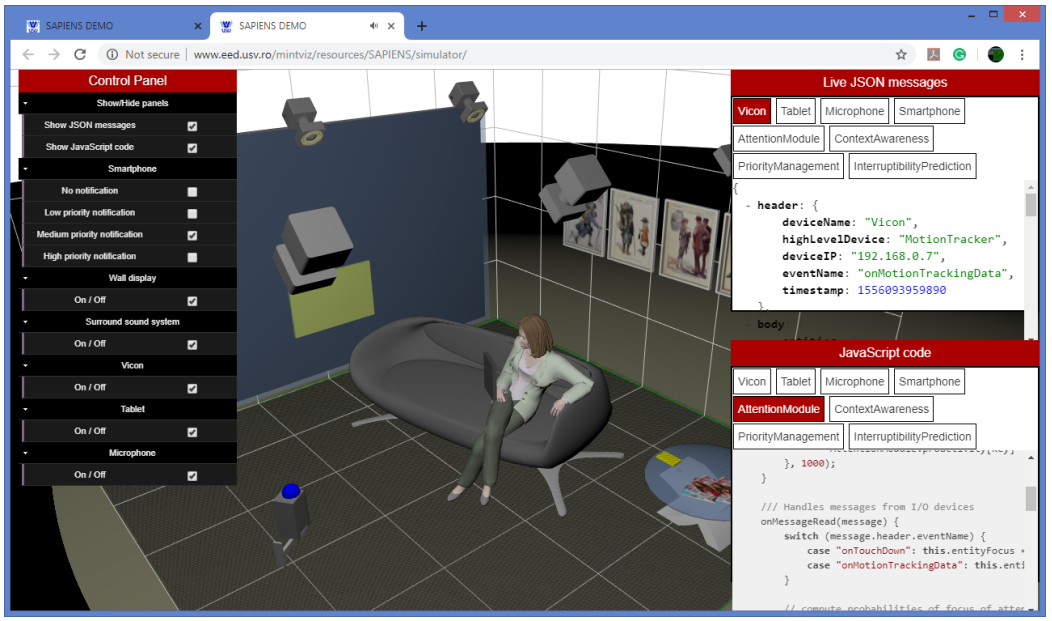

Fig. 6. Screenshot of our online simulator for the Sandra scenario running SAPIENS software modules and the EUPHORIA Engine. *Notes:* a Control Panel is available to the user on the left side of the screen to activate/deactivate devices and control the priority of notifications, while `JSON` messages and `JavaScript` code can be consulted in the panels shown on the right side of the screen.

`SurroundSound` and `WallDisplay`. This information is available to the PRIORITY-MANAGEMENT-MODULE to compile a list of devices based on their default priorities and the attention probabilities computed by the ATTENTION-DETECTION-MODULE. In our simulated scenario, low priority notifications from the `Smartphone` require just Sandra's peripheral attention and, consequently, they are forwarded to the `WallDisplay`. However, when a critical notification arrives, the INTERRUPTIBILITY-PREDICTION-MODULE uses its priority to direct it to the device that captures the focus of Sandra's attention based on the highest probability computed by the ATTENTION-DETECTION-MODULE. The CONTEXT-AWARENESS-MODULE is subscribed to events produced by `Vicon` and uses the corresponding messages to compile an ordered list of devices that are located in Sandra's vicinity.

4.3 Simulator

Our simulator is available at the web address http://www.eed.usv.ro/mintviz/resources/SAPIENS (see Figure 6 for a screenshot) and consists of the following user interface components:

(1) A *3-D graphical representation* of Sandra's environment, as depicted in Figure 1 from Section 1. The perspective of the scene can be adjusted by zooming in and out, panning, and rotating the point of view using mouse drag & drop and scroll actions.

(2) A *control panel* is available to users to activate/deactivate various functionalities and devices in the Sandra scenario (Figure 6, left). Check boxes are provided for each device to toggle operation on and off, *e.g.*, the `Tablet` can be deactivated with the immediate visible effect in the simulator that `JSON` messages are no longer produced by this device; deactivating the `SurroundSound` will stop the music from playing, etc. The control panel also enables simulation of priorities for the notifications received on Sandra's smartphone. This way, users

can observe the effect of low, medium, and high priority notifications on the JSON messages exchanged in the architecture and their corresponding impact on the output devices.

(3) *JSON messages* exchanged by the various components of the architecture are visible at all time (Figure 6, top right) and updated automatically. This feature enables users to observe live messages created by Producers (*e.g.*, touch input events produced by the Tablet, byte arrays generated by the Microphone object, or an array of location and orientation data for all registered devices provided by the Vicon object). The specialized components of Sapiens also produce and consume messages, *e.g.*, the message created by the Attention-Detection-Module to inform subscribed components about the device that captures Sandra's focus of attention. The fact that data are simulated for the input devices of our online environment (*e.g.*, the Tablet produces touch input events with screen coordinates generated at random) does not affect the implementation of the Sapiens specialized components.

(4) Actual *JavaScript code* implementing devices and specialized components is available for users to consult (Figure 6, bottom right). For example, Figure 7 illustrates an excerpt of the JavaScript implementation of the Attention-Detection-Module highlighting: (a) consumption of messages delivered through the Euphoria Engine from the Tablet, Smartphone, and Vicon devices (implemented under the onMessageRead(..) function); (b) computation of probabilities regarding Sandra's focus of attention for each device (implemented by the AttentionModule.probability[] array) using a technique based on rules; (c) computation of proactivity, expressed in seconds, for the device that has Sandra's focus of attention representing the likeliness of that device to keep Sandra engaged in the following seconds. We implemented proactivity using a function that decreases in geometric progression with common rate 0.9, *i.e.*, $\lambda(t) = \lambda(t-1) \cdot 0.9$, according to the notations from the previous section; and (d) production of a new message (corresponding to the onAttentionFocus event implemented under the onMessageWrite(..) function) to be delivered via the Euphoria Engine to other components of the architecture that subscribed to that event. For space concerns, we don't list in this article the source code of the other components, but instead we refer the interested reader to our online simulator.

5 CONCLUSION AND FUTURE WORK

We introduced Sapiens, our proposal for software architecture to support engineering of peripheral interactions in smart environments. Sapiens is multi-layer, modular, and flexible in terms of I/O devices. Future work will consider evaluation of practical implementations of Sapiens, such as the Sandra scenario discussed in this paper, but also other scenarios of different levels of complexity to understand aspects of technical performance, *e.g.*, the influence of the number of users, devices, and messages transferred through Sapiens on response time, by following the evaluation protcol from Schipor *et al.* [58]. Further exploration and engineering of the specialized components, such as more effective and generic ways to implement probabilistic reasoning and estimate hazard functions are also left for future work. At this moment, we deliver the description of Sapiens to the community as the first attempt to address engineering aspects of peripheral interaction through a dedicated software architecture with flexible layers and components. We also provide our specific implementation of the Sandra scenario in the form of an online simulation environment that practitioners can use to examine the dataflows of messages created, processed, and transmitted in Sapiens. Our software architecture has a dedicated web page, which invites constructive, fruitful discussion from the community towards consolidating a solid practice for engineering peripheral interactions in smart environments.

```
class AttentionModule {
  constructor(outputURL, inputURL, IP, events) {
    this.consumer = new EuphoriaConsumer(outputURL,events,this.onMessageRead.bind(this));
    this.producer = new EuphoriaProducer(inputURL,"AttentionModule","AttentionModule",IP,
        this.onMessageWrite,true);
    this.entityFocus = "";  // computes the entity that has the focus of attention
    this.message = this.producer.message; // message to be sent to the Engine
    this.onMessageWrite = this.onMessageWrite.bind(this);
    setInterval(() => {
        for (var key in AttentionModule.proactivity)
          AttentionModule.proactivity[key] *= 0.9; // non-linear decrease in time
      }, 1000);
  }

  /// Handles messages from I/O devices
  onMessageRead(message) {
    switch (message.header.eventName) {
      case "onTouchDown": this.entityFocus = message.header.deviceName; break;
      case "onMotionTrackingData": this.entityFocus = Vicon.getFocus(message); break;
    }

    // Compute probabilities of focus of attention for each device
    if (AttentionModule.probability[this.entityFocus] === undefined ||
        AttentionModule.probability[this.entityFocus] <
        AttentionModule.PROB_TRESHOLD)
        AttentionModule.probability[this.entityFocus] = AttentionModule.PROB_TRESHOLD;
    else AttentionModule.probability[this.entityFocus] *= 1.1; // non-linear increase

    // Normalize probabilities
    let sum = 0;
    for (var key in AttentionModule.probability) sum += AttentionModule.probability[key];
    for (var key in AttentionModule.probability) AttentionModule.probability[key] /= sum;

    // Set proactivity for the device that has the focus of attention
    AttentionModule.proactivity[this.entityFocus] = AttentionModule.PROACTIVITY_DEFAULT;

    // Send message to the Engine
    this.producer.onMessageWrite();
    this.onMessageWrite();
    this.producer.sendMessage();
  }

  /// Sends an attention message to the Engine
  onMessageWrite() {
    this.message.header.eventName = "onAttentionFocus";
    this.message.body.entityFocus = this.entityFocus;
    this.message.body.probability = AttentionModule.probability[this.entityFocus];
    this.message.body.proactivity = AttentionModule.proactivity[this.entityFocus];
  }
  ...
}

AttentionModule.probability = {}; // probability of attention, per device
AttentionModule.proactivity = {}; // proactivity, per device
AttentionModule.PROACTIVITY_DEFAULT = 10; // default proactivity, in seconds
AttentionModule.PROB_TRESHOLD = 1; // default (maximum) probability for devices having the
      user's focus of attention
```

Fig. 7. An excerpt of the JavaScript implementation of the Attention-Detection-Module, illustrating processing of messages received from input devices, computation of probabilities of attention for each device and of proactivity for the device that has the focus of attention, and production of an attention-related message transmitted to the Euphoria Engine.

ACKNOWLEDGMENTS

This work was supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI-UEFISCDI, project number PN-III-P3-3.1-PM-RO-CN-2018-0032 (3BM/2018), within PNCDI III.

REFERENCES

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999. Towards a better understanding of context and context-awareness. In *Proceedings of the International Symposium on Handheld and Ubiquitous Computing*. Springer, 304–307. https://doi.org/10.1007/3-540-48157-5_29

[2] Hamed S. Alavi and Pierre Dillenbourg. 2012. An Ambient Awareness Tool for Supporting Supervised Collaborative Problem Solving. *IEEE Trans. Learn. Technol.* 5, 3 (Jan. 2012), 264–274. https://doi.org/10.1109/TLT.2012.7

[3] Erik van Alphen and Saskia Bakker. 2016. Lernanto: Using an Ambient Display During Differentiated Instruction. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. ACM, New York, NY, USA, 2334–2340. https://doi.org/10.1145/2851581.2892524

[4] Jorge Alves Lino, Benjamin Salem, and Matthias Rauterberg. 2010. Responsive environments: User experiences for ambient intelligence. *Journal of Ambient Intelligence and Smart Environments* 2, 4 (2010), 347–367. https://doi.org/10.3233/AIS-2010-0080

[5] Juan Carlos Augusto. 2007. *Ambient Intelligence: The Confluence of Ubiquitous/Pervasive Computing and Artificial Intelligence*. Springer, London. https://doi.org/10.1007/978-1-84628-943-9_11

[6] Juan Carlos Augusto, H. Nakashima, and H. Aghajan. 2010. *Ambient Intelligence and Smart Environments: A State of the Art*. Springer Science & Business Media. https://doi.org/10.1007/978-0-387-93808-0_1

[7] Saskia Bakker. 2013. Design for peripheral interaction. *Eindhoven University of Technology* (2013). http://www.saskiabakker.com/PhDthesis_SaskiaBakker.pdf

[8] Saskia Bakker, Doris Hausen, and Ted Selker. 2016. Introduction: Framing peripheral interaction. In *Peripheral Interaction*. Springer, 1–10. https://doi.org/10.1007/978-3-319-29523-7_1

[9] Saskia Bakker, Elise Hoven, and Berry Eggen. 2015. Peripheral interaction: Characteristics and considerations. *Personal and Ubiquitous Computing* 19, 1 (2015), 239–254. https://doi.org/10.1007/s00779-014-0775-2

[10] Saskia Bakker and Karin Niemantsverdriet. 2016. The interaction-attention continuum: considering various levels of human attention in interaction design. *International Journal of Design* 10, 2 (2016), 1–14. http://ijdesign.org/index.php/IJDesign/article/view/2341/737

[11] Saskia Bakker, Elise van den Hoven, and Berry Eggen. 2013. FireFlies: Physical Peripheral Interaction Design for the Everyday Routine of Primary School Teachers. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction (TEI '13)*. ACM, New York, NY, USA, 57–64. https://doi.org/10.1145/2460625.2460634

[12] Saskia Bakker, Elise van den Hoven, Berry Eggen, and Kees Overbeeke. 2012. Exploring Peripheral Interaction Design for Primary School Teachers. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12)*. ACM, New York, NY, USA, 245–252. https://doi.org/10.1145/2148131.2148184

[13] Norbert Baumgartner, Wolfgang Gottesheim, Stefan Mitsch, Werner Retschitzegger, and Wieland Schwinger. 2010. Editorial: BeAware!-Situation Awareness, the Ontology-driven Way. *Data Knowl. Eng.* 69, 11 (Nov. 2010), 1181–1193. https://doi.org/10.1016/j.datak.2010.07.008

[14] John N.A. Brown. 2014. Unifying interaction across distributed controls in a smart environment using anthropology-based computing to make human-computer interaction "Calm". (2014). https://doi.org/10.13140/2.1.5166.4645

[15] Yaliang Chuang, Lin-Lin Chen, and Yoga Liu. 2018. Design Vocabulary for Human–IoT Systems Communication. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 274. https://doi.org/10.1145/3173574.3173848

[16] Diane Cook and Sajal Das. 2004. *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, New York, NY, USA.

[17] Diane J Cook, Juan C Augusto, and Vikramaditya R Jakkula. 2009. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing* 5, 4 (2009), 277–298. https://doi.org/10.1016/j.pmcj.2009.04.001

[18] Anind K. Dey. 2001. Understanding and Using Context. *Personal Ubiquitous Comput.* 5, 1 (Jan. 2001), 4–7. https://doi.org/10.1007/s007790170019

[19] Krista M. Dombroviak and Rajiv Ramnath. 2007. A Taxonomy of Mobile and Pervasive Applications. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)*. ACM, New York, NY, USA, 1609–1615. https://doi.org/10.1145/1244002.1244345

[20] Antonio Fernández-Caballero, Arturo Martínez-Rodrigo, José Manuel Pastor, José Carlos Castillo, Elena Lozano-Monasor, María T López, Roberto Zangróniz, José Miguel Latorre, and Alicia Fernández-Sotos. 2016. Smart environment architecture for emotion detection and regulation. *Journal of Biomedical Informatics* 64 (2016), 55–73. https://doi.org/10.1016/j.jbi.2016.09.015

[21] George H. Forman and John Zahorjan. 1994. The Challenges of Mobile Computing. *Computer* 27, 4 (April 1994), 38–47. https://doi.org/10.1109/2.274999

[22] Giancarlo Fortino, Daniele Parisi, Vincenzo Pirrone, and Giuseppe Di Fatta. 2014. BodyCloud: A SaaS Approach for Community Body Sensor Networks. *Future Gener. Comput. Syst.* 35 (June 2014), 62–79. https://doi.org/10.1016/j.future.2013.12.015

[23] Bogdan-Florin Gheran, Jean Vanderdonckt, and Radu-Daniel Vatavu. 2018. Gestures for Smart Rings: Empirical Results, Insights, and Design Implications. In *Proceedings of the 2018 Designing Interactive Systems Conference (DIS '18)*. ACM, New York, NY, USA, 623–635. https://doi.org/10.1145/3196709.3196741

[24] Giuseppe Ghiani, Marco Manca, and Fabio Paternò. 2015. Authoring context-dependent cross-device user interfaces based on trigger/action rules. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 313–322. https://doi.org/10.1145/2836041.2836073

[25] Milind S. Gide and Lina J. Karam. 2017. Computational Visual Attention Models. *Foundations and Trends® in Signal Processing* 10, 4 (2017), 347–427. https://doi.org/10.1561/2000000055

[26] Sylvain Giroux, Tatjana Leblanc, Abdenour Bouzouane, Bruno Bouchard, Hélène Pigot, and Jérémy Bauchet. 2009. The Praxis of Cognitive Assistance in Smart Homes. In *Ambient Intelligence and Smart Environments, volume 3: Behaviour Monitoring and Interpretation*. 183–211. http://dx.doi.org/10.3233/978-1-60750-048-3-183

[27] Christos Goumopoulos, Achilles Kameas, and Patras Hellas. 2009. Smart objects as components of ubicomp applications. *International Journal of Multimedia and Ubiquitous Engineering* 4, 3 (2009). http://www.sersc.org/journals/IJMUE/vol4_no3_2009/1.pdf

[28] Saul Greenberg, Nicolai Marquardt, Till Ballendat, Rob Diaz-Marino, and Miaosen Wang. 2011. Proxemic Interactions: The New Ubicomp? *Interactions* 18, 1 (Jan. 2011), 42–50. https://doi.org/10.1145/1897239.1897250

[29] Robert P. Hawkins, Suzanne Pingree, Jacqueline Hitchon, Barry Radler, Bradley W. Gorham, Leeann Kahlor, Eilleen Gilligan, Ronald C. Serlin, Toni Schmidt, Prathana Kannaovakun, and Gudbjorg Hildur Kolbeins. 2006. What Produces Television Attention and Attention Style? *Human Communication Research* 31, 1 (2006). https://doi.org/10.1111/j.1468-2958.2005.tb00868.x

[30] Marigo Heijboer, Elise Hoven, Bert Bongers, and Saskia Bakker. 2016. Facilitating Peripheral Interaction: Design and Evaluation of Peripheral Interaction for a Gesture-based Lighting Control with Multimodal Feedback. *Personal Ubiquitous Comput.* 20, 1 (Feb. 2016), 1–22. https://doi.org/10.1007/s00779-015-0893-5

[31] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. 2005. The Gator Tech Smart House: A programmable pervasive space. *Computer* 3 (2005), 50–60. https://doi.ieeecomputersociety.org/10.1109/MC.2005.107

[32] Laurent Itti. 2000. *Models of Bottom-up and Top-down Visual Attention*. Ph.D. Dissertation. Pasadena, CA, USA. Advisor(s) Koch, Christof. https://thesis.library.caltech.edu/4722/ AAI9972609.

[33] Jae Yeol Lee, Min Seok Kim, Dong Woo Seo, Sang Min Lee, and Jae Sung Kim. 2012. Smart and Space-aware Interactions Using Smartphones in a Shared Space. In *Proceedings of the 14th International Conference on Human-computer Interaction with Mobile Devices and Services Companion (MobileHCI '12)*. ACM, New York, NY, USA, 53–58. https://doi.org/10.1145/2371664.2371676

[34] Vicon Motion Systems Limited. 2019. Vicon Motion Capture. https://www.vicon.com/

[35] Jaime Lloret, Alejandro Canovas, Sandra Sendra, and Lorena Parra. 2015. A smart communication architecture for ambient assisted living. *IEEE Communications Magazine* 53, 1 (2015), 26–33. https://doi.org/10.1109/MCOM.2015.7010512

[36] Logbar. 2015. Ring ZERO | Shortcut anything. Retrieved April 18, 2019 from https://web.archive.org/web/20170511131824/http://ringzero.logbar.jp/

[37] J. Maisonnasse, N. Gourier, O. Brdiczka, P. Reignier, and J. L. Crowley. 2006. Detecting privacy in attention aware system. In *Proceedings of the 2nd IET International Conference on Intelligent Environments (IE '06)*, Vol. 2. 231–239. http://dx.doi.org/10.1049/cp:20060700

[38] Sathiya Kumaran Mani, Ramakrishnan Durairajan, Paul Barford, and Joel Sommers. 2018. An Architecture for IoT Clock Synchronization. In *Proceedings of the 8th International Conference on the Internet of Things (IOT '18)*. ACM, New York, NY, USA, Article 17, 8 pages. https://doi.org/10.1145/3277593.3277606

[39] Steve Mann. 1997. Wearable Computing: A First Step Toward Personal Imaging. *Computer* 30, 2 (Feb. 1997), 25–32. https://doi.org/10.1109/2.566147

[40] Alexander Mariakakis, Mayank Goel, Md Tanvir Islam Aumi, Shwetak N. Patel, and Jacob O. Wobbrock. 2015. SwitchBack: Using Focus and Saccade Tracking to Guide Users' Attention for Mobile Task Resumption. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 2953–2962. https://doi.org/10.1145/2702123.2702539

[41] Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. In *Proceedings of the 24th Annual ACM Symposium on User*

Interface Software and Technology (UIST '11). ACM, New York, NY, USA, 315–326. https://doi.org/10.1145/2047196.2047238

[42] Andrii Matviienko, Sebastian Horwege, Lennart Frick, Christoph Ressel, and Susanne Boll. 2016. CubeLendar: Design of a Tangible Interactive Event Awareness Cube. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. ACM, New York, NY, USA, 2601–2608. https://doi.org/10.1145/2851581.2892278

[43] Microsoft. 2017. Kinect for Windows 1.5, 1.6, 1.7, 1.8. Skeleton Class. https://msdn.microsoft.com/en-us/library/microsoft.kinect.skeleton.aspx

[44] Irina Mocanu, Ovidiu-Andrei Schpor, Bogdan Cramariuc, and Lucia Rusu. 2017. Mobile@Old: A Smart Home Platform for Enhancing the Elderly Mobility. *Advances in Electrical and Computer Engineering* 17, 4 (2017), 19–27. http://dx.doi.org/10.4316/AECE.2017.04003

[45] Meredith Ringel Morris. 2012. Web on the Wall: Insights from a Multimodal Interaction Elicitation Study. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces (ITS '12)*. ACM, New York, NY, USA, 95–104. https://doi.org/10.1145/2396636.2396651

[46] MyGaze. 2019. myGaze eye tracker. http://www.mygaze.com/products/mygaze-eye-tracker/

[47] Alfredo Navarra, Cristina M. Pinotti, Mario Francesco, and Sajal K. Das. 2015. Interference-free Scheduling with Minimum Latency in Cluster-based Wireless Sensor Networks. *Wirel. Netw.* 21, 7 (Oct. 2015), 2395–2411. https://doi.org/10.1007/s11276-015-0925-0

[48] Michael Nebeling, Elena Teunissen, Maria Husmann, and Moira C. Norrie. 2014. XDKinect: Development Framework for Cross-device Interaction Using Kinect. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '14)*. ACM, New York, NY, USA, 65–74. https://doi.org/10.1145/2607023.2607024

[49] North. 2019. Getting started with your Myo armband. Retrieved April 18, 2019 from https://support.getmyo.com/hc/en-us/articles/203398347-Getting-started-with-your-Myo-armband

[50] Bogdan Pogorelc, Artur Lugmayr, Bjorn Stockleben, Radu-Daniel Vatavu, Nina Tahmasebi, Estefania Serral, Emilija Stojmenova, Bojan Imperl, Thomas Risse, Gideon Zenz, and Matjaz Gams. 2013. Ambient Bloom: New Business, Content, Design and Models to Increase the Semantic Ambient Media Experience. *Multimedia Tools and Applications* 66, 1 (2013), 7–32. http://dx.doi.org/10.1007/s11042-012-1228-4

[51] Bogdan Pogorelc, Radu-Daniel Vatavu, Artur Lugmayr, Björn Stockleben, Thomas Risse, Juha Kaario, Estefania Constanza Lomonaco, and Matjaž Gams. 2012. Semantic ambient media: From ambient advertising to ambient-assisted living. *Multimedia Tools and Applications* 58, 2 (May 2012), 399–425. https://doi.org/10.1007/s11042-011-0917-8

[52] Irina Popovici, Ovidiu-Andrei Schipor, and Radu-Daniel Vatavu. 2019. Hover: Exploring cognitive maps and mid-air pointing for television control. *International Journal of Human-Computer Studies* 129 (2019), 95–107. https://doi.org/10.1016/j.ijhcs.2019.03.012

[53] Robbie Schaefer and Wolfgang Mueller. 2003. Multimodal interactive user interfaces for mobile multi-device environments. In *Workshop" Multi-Device Interfaces for Ubiquitous Peripheral Interaction*.

[54] Ovidiu-Andrei Schipor, Stefan-Gheorghe Pentiuc, and Maria-Doina Schipor. 2011. Towards a multimodal emotion recognition framework to be integrated in a Computer Based Speech Therapy System. In *2011 6th Conference on Speech Technology and Human-Computer Dialogue (SpeD)*. IEEE, 1–6. https://doi.org/10.1109/SPED.2011.5940727

[55] Ovidiu-Andrei Schipor, Stefan-Gheorghe Pentiuc, and Maria-Doina Schipor. 2012. Toward Automatic Recognition of Children's Affective State Using Physiological Parameters and Fuzzy Model of Emotions. *Advances in Electrical and Computer Engineering* 12, 2 (2012), 47–50. https://doi.org/10.4316/aece.2012.02008

[56] Ovidiu-Andrei Schipor, Doina-Maria Schipor, Emilia Crismariu, and Stefan-Gheorghe Pentiuc. 2011. Finding key emotional states to be recognized in a computer based speech therapy system. *Procedia-Social and Behavioral Sciences* 30 (2011), 1177–1182. https://doi.org/10.1016/j.sbspro.2011.10.229

[57] Ovidiu-Andrei Schipor and Radu-Daniel Vatavu. 2018. Invisible, Inaudible, and Impalpable: Users' Preferences and Memory Performance for Digital Content in Thin Air. *IEEE Pervasive Computing* 17, 4 (2018), 76–85. https://doi.org/10.1109/MPRV.2018.2873856

[58] Ovidiu-Andrei Schipor, Radu-Daniel Vatavu, and Jean Vanderdonckt. 2019. Euphoria: A Scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments. *Information and Software Technology* 109 (May 2019), 43–59. https://doi.org/10.1016/j.infsof.2019.01.006

[59] Teddy Seyed, Alaa Azazi, Edwin Chan, Yuxi Wang, and Frank Maurer. 2015. Sod-toolkit: A toolkit for interactively prototyping and developing multi-sensor, multi-device environments. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces*. ACM, 171–180. https://doi.org/10.1145/2817721.2817750

[60] StackOverflow. 2018. Stack Overflow Developer Survey 2018. https://insights.stackoverflow.com/survey/2018#technology-programming-languages

[61] Cristian Andy Tănase, Radu-Daniel Vatavu, Ştefan-Gheorghe Pentiuc, and Adrian Graur. 2008. Detecting and tracking multiple users in the proximity of interactive tabletops. *Advances in Electrical and Computer Engineering* 8, 15 (2008),

61–64. http://dx.doi.org/10.4316/AECE.2008.02011

[62] Peter Tolmie, James Pycock, Tim Diggins, Allan MacLean, and Alain Karsenty. 2002. Unremarkable Computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*. ACM, New York, NY, USA, 399–406. https://doi.org/10.1145/503376.503448

[63] Sandra Trullemans, Lars Van Holsbeeke, and Beat Signer. 2017. The context modelling toolkit: A unified multi-layered context modelling approach. *Proceedings of the ACM on Human-Computer Interaction* 1, EICS (2017), 8. https://doi.org/10.1145/3095810

[64] Radu-Daniel Vatavu. 2012. Presence Bubbles: Supporting and Enhancing Human-Human Interaction with Ambient Media. *Multimedia Tools and Applications* 58, 2 (2012), 371–383. http://dx.doi.org/10.1007/s11042-010-0674-0

[65] Radu-Daniel Vatavu. 2012. User-defined Gestures for Free-hand TV Control. In *Proceedings of the 10th European Conference on Interactive TV and Video (EuroITV '12)*. ACM, New York, NY, USA, 45–48. https://doi.org/10.1145/2325616.2325626

[66] Radu-Daniel Vatavu. 2013. On Designing Interactivity Awareness for Ambient Displays. *Multimedia Tools and Applications* 66, 1 (2013), 59–80. https://doi.org/10.1007/s11042-012-1140-y

[67] Radu-Daniel Vatavu. 2015. Audience Silhouettes: Peripheral Awareness of Synchronous Audience Kinesics for Social Television. In *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video (TVX '15)*. ACM, New York, NY, USA, 13–22. https://doi.org/10.1145/2745197.2745207

[68] Radu-Daniel Vatavu. 2017. Smart-Pockets: Body-Deictic Gestures for Fast Access to Personal Data during Ambient Interactions. *International Journal of Human-Computer Studies* 103, C (July 2017), 1–21. https://doi.org/10.1016/j.ijhcs.2017.01.005

[69] Radu-Daniel Vatavu and Ionut-Alexandru Zaiti. 2014. Leap Gestures for TV: Insights from an Elicitation Study. In *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video (TVX '14)*. ACM, New York, NY, USA, 131–138. https://doi.org/10.1145/2602299.2602316

[70] Jo Vermeulen, Steven Houben, and Nicolai Marquardt. 2016. Fluent Transitions Between Focused and Peripheral Interaction in Proxemic Interactions. In *Peripheral Interaction*. Springer, 137–163. https://doi.org/10.1007/978-3-319-29523-7_7

[71] Daniel Vogel and Ravin Balakrishnan. 2004. Interactive Public Ambient Displays: Transitioning from Implicit to Explicit, Public to Personal, Interaction with Multiple Users. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/1029632.1029656

[72] Emily Vraga, Leticia Bode, and Sonya Troller-Renfree. 2016. Beyond self-reports: Using eye tracking to measure topic and style differences in attention to social media content. *Communication Methods and Measures* 10, 2-3 (2016), 149–164. https://doi.org/10.1080/19312458.2016.1150443

[73] Florian Wahl, Martin Freund, and Oliver Amft. 2015. WISEglass: Smart Eyeglasses Recognising Context. In *Proceedings of the 10th EAI International Conference on Body Area Networks (BodyNets '15)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 11–17. https://doi.org/10.4108/eai.28-9-2015.2261470

[74] Miaosen Wang, Sebastian Boring, and Saul Greenberg. 2012. Proxemic Peddler: A Public Advertising Display That Captures and Preserves the Attention of a Passerby. In *Proceedings of the 2012 International Symposium on Pervasive Displays (PerDis '12)*. ACM, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/2307798.2307801

[75] Mark Weiser. 1999. The Computer for the 21st Century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3 (July 1999), 3–11. https://doi.org/10.1145/329124.329126

[76] Mark Weiser and John Seely Brown. 1997. The coming age of calm technology. In *Beyond calculation*. Springer, 75–85. https://doi.org/10.1007/978-1-4612-0685-9_6

[77] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. 2009. User-defined Gestures for Surface Computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1083–1092. https://doi.org/10.1145/1518701.1518866

Received February 2019; revised March 2019; accepted April 2019