SUMMARY:

Design report for in-vehicle augmented reality software architecture

Complex project:

PN-III-P1-1.2-PCCDI-2017-0917

Component project:

P2 – Efficient communication based on smart devices in interactive in-vehicle augmented reality scenarios

Partners:

Ovidius University of Constanța

Ștefan cel Mare University of Suceava

Authors:

Ovidius University of Constanța

Prof.univ.dr. Dorin Mircea POPOVICI
Conf.univ.dr. Dragoş-Florin SBURLAN
Conf.univ.dr. Crenguța Mădălina PUCHIANU
Lect.univ.dr. Elena BĂUTU

Ştefan cel Mare University of Suceava

Prof.dr.ing. Radu-Daniel VATAVU
Prof.dr.ing. Ştefan-Gheorghe PENTIUC
Conf. univ. dr.ing. Ovidiu-Andrei SCHIPOR

Table of Content

1. Introduction

The 2018 implementation stage for the component project P2 consisted in developing a software infrastructure for augmented reality in-vehicle systems, in accordance with the Activity 1.2. "Developing of a software infrastructure for augmented reality in-vehicle systems", of the development plan of the component projects – a type A2 activity (industrial research). The main goal behind the development of the software architecture is to offer the support necessary for the design, implementation and evaluation of the modules and components planned for the next stages of the component project P2.

The technical and scientific activities in the 2018 project stage consisted in:

- Designing a software architecture based on event processing called Euphoria (Event-based Unified Platform for HeteRogeneous and Aynchronous Interactions), that has a high level of generality, in order to easily integrate various types of input and output devices. Accordingly, we prepared the technical design documentation for the software architecture.
- Implementation of the software architecture Euphoria, using industry-standard and validated web technologies.
- Designing and deploying software components that send / receive messages to / from the Euphoria server.
- Evaluating the technical performance of the Euphoria software architecture in terms of event processing speed and scalability. Accordingly, we prepared a documentation for testing the architecture.
- Preparation, writing and submitting for evaluation and publication scientific papers, on the theme of component project P2.

The current document represents the summary of the design report of stage 1 – 2018.

2. Design and implementation of the Euphoria architecture

This report summarizes the design and implementation details for **Euphoria (Event-based Unified Platform for HeteRogeneous and Aynchronous Interactions),** a novel software architecture design and implementation that enables prototyping and evaluation of flexible, asynchronous interactions between users, personal devices, and public installations, systems, and services within smart environments of all kinds, such as the augmented reality in-vehicle systems addressed by component project P2.

Euphoria implements the production, transmission, processing and consumption of events using protocols and web technologies well known in the industrial sector, such as HTTP, WebSockets, node.js, JSON, etc. These are natively supported by a wide variety of interconnected devices, such as PCs, tablets and smart watches, etc.

Euphoria is available online at the permanent web address http://www.eed.usv.ro/mintviz/resources/Euphoria

2.1. Design requirements and performance criteria

We designed the Euphoria architecture by considering a series of performance criteria, described in detail in this section. To properly position the Euphoria software architecture within the specific literature, we present a comparative analysis with respect to existing similar software architectures.

The Euphoria software architecture was designed to easily integrate devices and software apps within an intelligent environment, consisting in a variety of consumers and producers. The specific goal of the architecture is to be applied for scenarios involving user interactions in smart connected cars. Hence, we adopted several by adopting several handling techniques and quality properties; see Figure 1 for a visual overview.

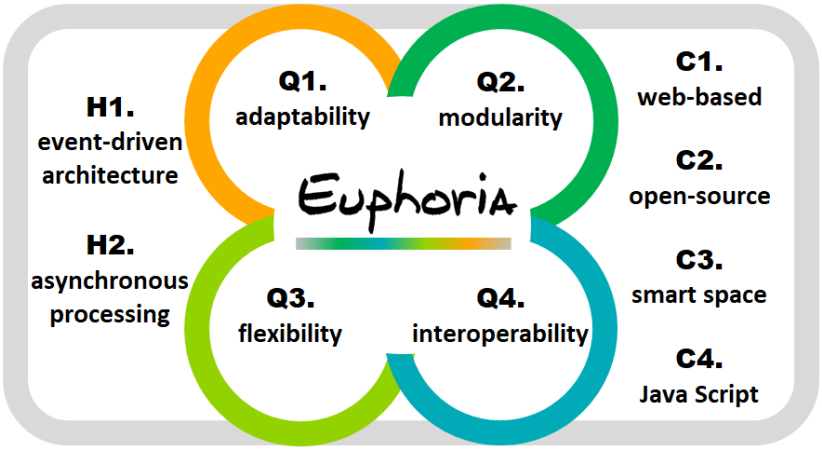

Figure 1. Visual overview of the two handling techniques (H 1 and H 2), the four quality properties (Q1 , Q2 , Q3 , and Q4), and the four contextual properties (C1 , C2 , C3 , and C4) adopted for the design of the Euphoria software architecture.

2.2. Designing the Euphoria software architecture

This section presents the model of the software architecture Euphoria. Figure 2 illustrated the design of Euphoria having as main components: producers, transmitters, processing engine, receivers and consumers, according to the model of event-based architectures [19]. We designed Euphoria with a strong separation between its functional blocks (Q2) and with a reduced size of its adaptation layers (Q1).

In accordance with de design principles H1 [18, 19, 20], the information about the events triggered by the producers are collected, packaged in messages and exposed by the emitters to the next layer. Subsequently, the messages are processed and shipped by the processing engine to the receivers and consumers, who will interpret those messages according to their internal logic and take appropriate action within the intelligent environment.

For example, an event detected by a smart watch worn by the driver of a smart connected car will lead to the selection of an element on the screen. A message sent through Euphoria consists in a header with metadata (e.g., the name of the device, the IP address, the name of the event) and a message body, which contains the data that is relevant for the event. Figure 2 illustrates four examples of messages of various complexity, produced by a bracelet, a smartphone, and two motion detection sensors.

The producer represents the software component that generates events: a (slight) alteration of its state is notified to the software architecture as an event. Producers are instantiated by the input devices, (for example, motion sensors, intelligent mobile devices, portable devices) or software services (for example, a gesture command that has been detected for a specific user by a gesture recognition service on the web).

For the producers that are instances of input devices, the events refer to any physical alteration of the states of these devices that is relevant for the application, such as pressing a button or a change in the wrist orientation detected by a smart watch. Similarly, software events can be produced in relation to changes in the state of a given application.

Software events prove to be useful for debugging, simulation, or integration of the complex logic provided by third-party libraries and services, such as those available on the web. Another characteristic of event-based architectures is that distinct layers do not know the operating details of the other layers of the architecture (Q2).

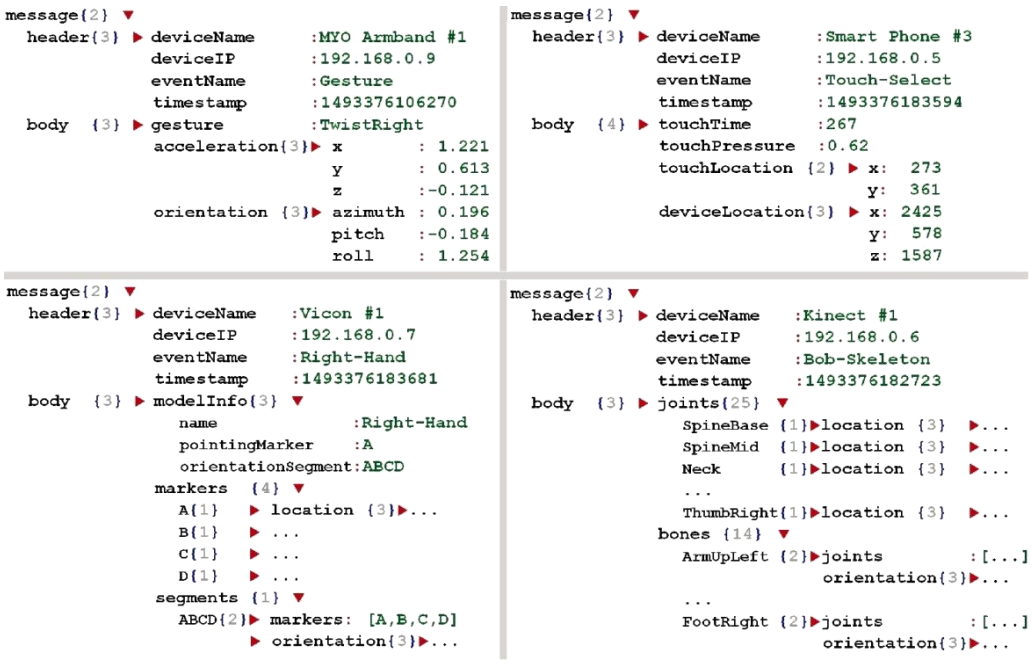

Figure 2. Examples of JSON messages of various complexity: a message containing acceleration and orientation data collected from a Myo armband (top left); touch input data reported by a smartphone (top right); motion data tracked by a Vicon system (bottom left); and a message containing a Microsoft Kinect skeleton (bottom right).

For example, it is not important for producers to know which components will process and consume the events that they produce. This feature facilitates the integration of devices and new software applications in Euphoria, including future ones, such as mobile augmented reality devices [61], without any changes to the main components of the Euphoria architecture.

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

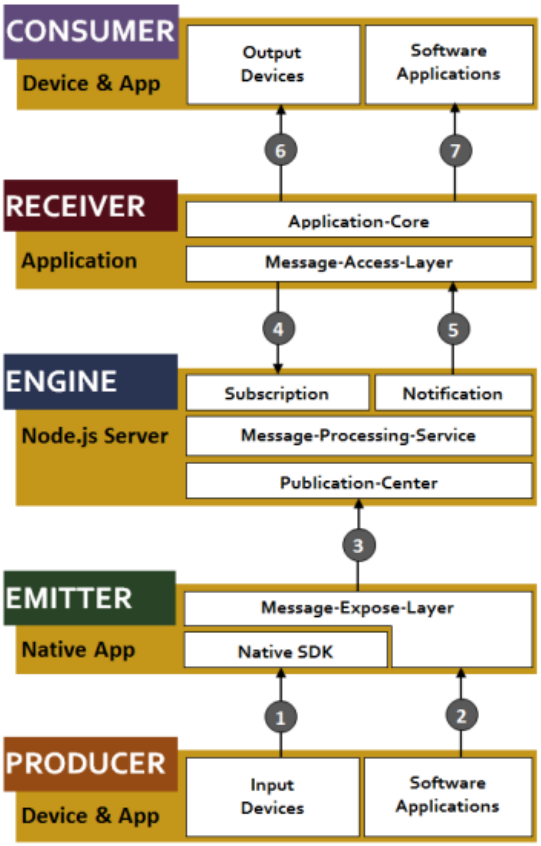

Figure 3. The layered architecture of Euphoria, an event-based software architecture for designing interactions in smart environments. Euphoria consists of event producers, emitters, a processing engine, and event receivers and consumers. Numbers ❶ to ❼ represent data flows that are discussed in the text.

The Emitter collects event data and packs them in the form of messages that are delivered to the higher layers of the architecture. A message consists of a header with metadata (i.e., the event type and the identification of the producer) and a message body with details about the change of state (e.g., the button that was pressed, the type of gesture performed by the user, new position coordinates from a motion sensor, etc.). While the header of the message has a fixed structure, its body depends on the type of device and the event that was produced.

In most cases, the Message-Expose-Layer (see Figure 3) needs to be implemented in native code, although some SDKs or software applications may already offer higher-level interfacing options for a given input device.

Emitters (producers) and receivers (consumers) represent the interface between the Euphoria kernel and the wide range of devices and software apps that we wish to connect. In order to send or receive a message, a software conversion is needed, between a model for the producer/consumer and a generic model, specific to Euphoria. The use of robust and optimal technologies, like HTTP or JSON, leads to an adaptive layer as efficient as possible, while maintaining flexibility and interoperability. In fact, emitters and receivers are the only two levels of the architecture that need to be implemented by the developers.

Since the emitter component is necessary to integrate each input device in the apps that will reside in the smart environment, we designed it in a general and reusable manner, according to quality properties Q3 and Q4. Basically, the emitter consists only of the code needed to encapsulate the specific properties in a standard JSON message and to send this message through an HTTP request. Feeding our architecture with an event is just as simple as an HTTP request to a URL: the query string represents the message of the event.

The processing engine is the essential component of the software architecture Euphoria. Its main responsibility is to receive messages from Emitters (see dataflow ❸ in Figure 3) and to dispatch them to the appropriate Consumers that have registered to receive those specific types of events. In our implementation, the Processing-Engine is run by a node.js web server. The event messages collected by the Publication-Center are unpacked by the Message-Processing-Service; see Figure 3.

Before receiving a notification, a consumer must register, following a standard protocol. As a part of this protocol, the consumer must specify the type of the events it will be able to receive, as well as the corresponding producers. Once the registration is finalized, the processing engine will automatically notify the consumer with respect to those events that meet the requirements agreed upon during the registration.

To perform 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 [62, 63].

The Message-Processing-Service (Figure 3) routes the messages to the appropriate Consumers and converts the data in the JSON (JavaScript Object Notation) interchangeable format (our quality property Q4). The processing engine does not know

the internal structure of producers or consumers. This abstraction supports both scalability (quality property 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 use them for its own business logic. The receiver establishes an WebSockets connection with the processing engine, by means of the Message-Access-Layer (see dataflows ❹ and ❺ in Figure 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 of Euphoria is to maintain a very thin layer between a Receiver and the Processing-Engine so that any existing software application can integrate with Euphoria with a minimum coding effort (quality properties Q1 and Q2). To this end, we decided to implement the Message-Access-Layer using the industry-mature technology of WebSockets, enabling thus real-time and asynchronous data transfer (see handling technique 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 Figure 3). This tier consists in output devices and software applications that users interact with directly. It is the output interface between Euphoria and the outer environment. The implementation details of Consumers are locked within the Receiver layer to make the Engine as decoupled as possible.

3. Design and deployment of software components that communicate with the Euphoria architecture

The event producer and consumer apps, *GenericProducer* and *GenericConsumer*, were implemented as prototypes and tested in this stage of the project. Using these prototypes, the driver may choose to receive messages that contain information regarding the weather. Also, he may choose to receive or block notification received

from third party applications, such as Facebook, Gmail, WhatsApp, or any other notification automatically received by his mobile device (e.g. automatic updates for installed software). This way, the driver is notified only with information that is important to him, his attention being distracted as little as possible from his driving activity.

The *GenericProducer* app includes, at this moment, 3 types of content producers:

- a generic content producer, entitled *GenericComponent*. It is customizable, and it delivers customizable notifications / messages at dynamically established time intervals. This producer has proven useful in testing the application in terms of management of the time and platform resources.

- a content producer called *NotificationComponent*, which receives all notifications from the mobile device as soon as they appear. It packs them in a unified manner and sends them as a JSON-like message to the Euphoria server.

- a content producer entitled *WeatherComponent*, which generates messages containing information about the weather at the location of the mobile device. This component reads the location of the mobile device of the driver; after that, it makes a connection to a weather service available online. Finally, the component produces regular notifications, at customizable time intervals, with information about the weather at the current location.

The *GenericConsumer* app receives JSON objects from the Euphoria architecture. It displays information about them on the consumer's screen. Now, the purpose of display was to test the reliability of such an approach. In order to display the information from the producer, we first took into account the visibility of the information displayed under different environmental conditions. Another important goal was reducing to a minimum the disturbance of the attention of the traffic participants in general, and of the driver, in particular, brought on by displaying the information.

Each app was designed using the event driven methodology, based on the broadcasting version of the design pattern Observer [66]. We applied this style in order to obtain independent components that asynchronously process the data.

Each event producer/consumer was encapsulated in a package that was designed using the object-oriented methodology, using UML (Unified Modeling Language)[1] [64]:

- from the point of view of the interactions between objects – we designed a UML sequence diagram
- from a static (structural) point of view – we designed a UML class diagram that depicts the classes and interfaces in each package, and the relations among them.

In designing these UML diagrams, we applied general GRASP software design patterns, such as Information Expert, Creator, Low Coupling, High Cohesion, Pure Fabrication, Indirection [67]. These design patterns provide a guarantee that we have obtained a quality architecture that applies basic design principles, such as poor coupling of classes or high-class cohesion.

4. References

[1] S. Poslad, Ubiquitous computing: smart devices, environments and interactions, John Wiley & Sons, 2011. URL http://dx.doi.org/10.1002/9780470779446

[2] P. Friess, Internet of things: converging technologies for smart environments and integrated ecosystems, River Publishers, 2013.

[3] O.-A. Schipor, W. Wu, W.-T. Tsai, R.-D. Vatavu, Software architecture design for spatially-indexed media in smart environments, Advances in Electrical and Computer Engineering 17 (2) (2017) 17–22. URL http://dx.doi.org/10.4316/AECE.2017.02003

[4] 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. URL https://doi.org/10.1007/978-3-642-34898-3_11

[5] Y. Lou, W. Wu, R.-D. Vatavu, W.-T. Tsai, Personalized gesture interactions for cyber-physical smart-home environments, Science China Information Sciences 60 (7) (2017) 072104. URL https://doi.org/10.1007/s11432-015-1014-7

[6] B.-F. Gheran, J. Vanderdonckt, R.-D. Vatavu, Gestures for smart rings: Empirical results, insights, and design implications, in: Proceedings of the 2018 Designing

[1]UML is a standard description of the models created in the development phases of a software system: software requirements analysis and design [64]

Interactive Systems Conference, DIS '18, ACM, New York, NY, USA, 2018, pp. 623–635. URL http://doi.acm.org/10.1145/3196709.3196741

[7] G. Bailly, J. Muller, 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, CHI '12, ACM, New York, NY, USA, 2012, pp. 1239–1248. URL http://doi.acm.org/10.1145/2207676.2208576

[8] 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. URL http://dx.doi.org/10.1016/j.future.2013.12.015

[9] P.-V. Cioata, 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, IUI '18 Companion, ACM, New York, NY, USA, 2018, pp. 60:1–60:2. URL http://doi.acm.org/10.1145/3180308.3180369

[10] 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, CHI '14, ACM, New York, NY, USA, 2014, pp. 159–168. URL http://doi.acm.org/10.1145/2556288.2556955

[11] 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, UIST '15, ACM, New York, NY, USA, 2015, pp. 457–466. URL http://doi.acm.org/10.1145/2807442.2807499

[12] 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. URL https://doi.org/10.1007/978-3-319-00491-4_1

[13] 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, EICS '14, ACM, New York, NY, USA, 2014, pp. 65–74. URL http://doi.acm.org/10.1145/2607023.2607024

[14] C. Roda, A. Rodr´ıguez, E. Navarro, V. Lopez-Jaquero, P. Gonzalez, 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. URL https://doi.org/10.1007/978-3-319-40159-1_26

[15] R. D. Hill, Event-response systems: A technique for specifying multi-threaded dialogues, SIGCHI Bull. 18 (4) (1986) 241–248. URL http://doi.acm.org/10.1145/1165387.275637

[16] ISO, Iec 25000 software and system engineering–software product quality require-ments and evaluation (square)–guide to square, International Organization for Standarization. URL https://www.iso.org/standard/35683.html

[17] F. Schneider, B. Berenbach, A literature survey on international standards for systems requirements engineering, Procedia Computer Science 16 (2013) 796–805. URL https://doi.org/10.1016/j.procs.2013.01.083

[18] E. Yourdon, L. L. Constantine, Structured design: Fundamentals of a discipline of computer program and systems design, Prentice-Hall, Inc., 1979.

[19] 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 process-ing systems, IBM Redguide publicationdoi:10.1.1.454.8442.

[20] K. M. Chandy, Event-driven applications: Costs, benefits and design approaches, Gartner Application Integration and Web Services Summit 2006.

[21] 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 Systems Journal 10 (3) (2016) 1137–1146. URL https://www.doi.org/10.1109/JSYST.2014.2302750

[22] 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. URL http://dx.doi.org/10.1147/sj.384.0693

[23] 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, Ch. Ambient Intelligence, pp. 235–250. URL http://dl.acm.org/citation.cfm?id=504949.504964

[24] M. Kuniavsky, Smart Things: Ubiquitous computing user experience design, Mor-gan Kaufmann, Burlington, MA, USA, 2010. URL http://dx.doi.org/10.1016/C2009-0-20057-2

[25] D. Cook, S. K. Das, Smart Environments: Technology, Protocols and Applications, Wiley-Interscience, 2004. URL http://dx.doi.org/10.1002/047168659X

[26] 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. URL https://doi.org/10.1109/FRUCT.2014.6872437

[27] R.-D. Vatavu, A. Mossel, C. Schonauer, 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, MobileHCI '16, ACM, New York, NY, USA, 2016, pp. 217– URL http://doi.acm.org/10.1145/2935334.2935364226.

[28] 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 - Vol-ume 9677, ICOST 2016, Springer-Verlag, Berlin, Heidelberg, 2016, pp. 464–476. URL https://doi.org/10.1007/978-3-319-39601-9_42

[29] M. J. Deen, Information and communications technologies for elderly ubiquitous healthcare in a smart home, Personal Ubiquitous Comput. 19 (3-4) (2015) 573–599. URL http://dx.doi.org/10.1007/s00779-015-0856-x

[30] 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. URL https://doi.org/10.1109/ICS.2016.0044

[31] A. Pfister, A. M. West, S. Bronner, J. A. Noah, Comparative abilities of microsoft kinect and vicon 3d motion capture for gait analysis, Journal of medical engineering & technology 38 (5) (2014) 274–280. URL https://doi.org/10.3109/03091902.2014.909540

[32] M. Lui, J. D. Slotta, Immersive simulations for smart classrooms: exploring evolutionary concepts in secondary science, Technology, Pedagogy and Education 23 (1) (2014) 57–80. URL https://doi.org/10.1080/1475939X.2013.838452

[33] B. Brown, K. O'Hara, T. Kindberg, A. Williams, Crowd computer interaction, in: CHI '09 Extended Abstracts on Human Factors in Computing Systems, CHI EA '09, ACM, New York, NY, USA, 2009, pp. 4755–4758. URL http://doi.acm.org/10.1145/1520340.1520733

[34] 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, CHI EA '18, ACM, New York, NY, USA, 2018, pp. LBW032:1–LBW032:6. URL http://doi.acm.org/10.1145/3170427.3188465

[35] 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. URL https://doi.org/10.1109/RCIS.2013.6577709

[36] J. Vanderdonckt, Accessing guidelines information with sierraProc. of IFIP Conf. on Human-Computer Interaction, Vol. 95, 2016, 311–316. URL https://doi.org/10.1007/978-1-5041-2896-4_52

[37] 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. URL https://doi.org/10.1007/978-3-319-40180-5_13

[38] O. A. Schipor, et al., 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. URL http://arxiv.org/abs/1405.7796

[39] O.-I. Gherman, O.-A. Schipor, B.-F. Gheran, Verge: A system for collecting voice, eye gaze, gesture, and eeg data for experimental studies, in: Proceedings of the 14th International Conference on Development and Application Systems, DAS ’18, 2018, pp. 150–155. URL http://dx.doi.org/10.1109/DAAS.2018.8396088

[40] K. Bahreini, R. Nadolski, W. Westera, Towards multimodal emotion recognition in e-learning environments, Interactive Learning Environments 24 (3) (2016) 590– 605. URL https://doi.org/10.1080/10494820.2014.908927

[41] 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. URL https://doi.org/10.1109/IIKI.2015.69

[42] D. J. Cook, N. C. Krishnan, P. Rashidi, Activity discovery and activity recognition: A new partnership, IEEE transactions on cybernetics 43 (3) (2013) 820–828. URL https://doi.org/10.1109/TSMCB.2012.2216873

[43] S. S. Rautaray, A. Agrawal, Vision based hand gesture recognition for human computer interaction: A survey, Artif. Intell. Rev. 43 (1) (2015) 1–54. URL http://dx.doi.org/10.1007/s10462-012-9356-9

[44] M. R. Abid, E. M. Petriu, E. Amjadian, Dynamic sign language recognition for smart home interactive application using stochastic linear formal grammar, IEEE Transactions on Instrumentation and Measurement 64 (3) (2015) 596–605. URL https://doi.org/10.1109/TIM.2014.2351331

[45] O. Schipor, S. Pentiuc, M. Schipor, The utilization of feedback and emotion recog-nition in computer based speech therapy system, Elektronika ir Elektrotechnika 109 (3) (2011) 101–104. URL http://dx.doi.org/10.5755/j01.eee.109.3.181

[46] J. K. Zao, T. T. Gan, C. K. You, S. J. R. M´endez, C. E. Chung, Y. Te Wang, T. Mullen, T. P. Jung, Augmented brain computer interaction based on fog com-puting and linked

data, in: Intelligent Environments (IE), 2014 International Con-ference on, IEEE, 2014, pp. 374–377. URL https://doi.org/10.1109/IE.2014.54

[47] R.-D. Vatavu, Smart-pockets: Body-deictic gestures for fast access to personal data during ambient interactions, International Journal of HumanComputer Studies 103 (2017) 1–21. URL http://dx.doi.org/10.1016/j.ijhcs.2017.01.005

[48] 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, CHI EA '14, ACM, New York, NY, USA, 2014, pp. 2305– 2310. URL http://doi.acm.org/10.1145/2559206.2581143

[49] 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, CHI EA '15, ACM, New York, NY, USA, 2015, pp. 103–108. URL http://doi.acm.org/10.1145/2702613.2726961

[50] C. Goumopoulos, A. Kameas, P. Hellas, Smart objects as components of ubi-comp applications, International Journal of Multimedia and Ubiquitous Engineer-ing 4 (3). URL http://www.sersc.org/journals/IJMUE/vol4_no3_2009/1.pdf

[51] 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. URL http://dx.doi.org/10.1093/comjnl/bxq019

[52] 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. URL https://doi.org/10.1002/spe.998

[53] G. Fortino, A. Guerrieri, G. M. P. O'Hare, A. Ruzzelli, A flexible building manage-ment framework based on wireless sensor and actuator networks, J. Netw. Comput. Appl. 35 (6) (2012) 1934–1952. URL http://dx.doi.org/10.1016/j.jnca.2012.07.016

[54] 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 sup-porting future internet applications, in: Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on, IEEE, 2014, pp. 792–799. URL https://doi.org/10.1109/UCC.2014.129

[55] F. Lab, Fiware lab (2018). URL http://status.lab.fiware.org/

[56] E. Hall, The Hidden Dimension (Anchor Books a Doubleday Anchor Book), An-chor, 1966. URL http://www.worldcat.org/isbn/0385084765

[57] N. Marquardt, R. Diaz-Marino, S. Boring, S. Greenberg, The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies, in: Pro-ceedings

of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST '11, ACM, New York, NY, USA, 2011, pp. 315–326. URL http://doi.acm.org/10.1145/2047196.2047238

[58] A. P. Volpentesta, A framework for human interaction with mobiquitous services in a smart environment, Comput. Hum. Behav. 50 (C) (2015) 177–185. URL http://dx.doi.org/10.1016/j.chb.2015.04.003

[59] D. Ledo, S. Greenberg, N. Marquardt, S. Boring, Proxemic-aware controls: De-signing remote controls for ubiquitous computing ecologies, in: Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile De-vices and Services, MobileHCI '15, ACM, New York, NY, USA, 2015, pp. 187–198. URL http://doi.acm.org/10.1145/2785830.2785871

[60] 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, Vol. 2, " Carol I" National Defence University, 2017, p. 487. URL https://doi.org/10.12753/2066-026x-17-154

[61] 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. URL https://doi.org/10.1109/ACCESS.2017.2698164

[62] V. Wang, F. Salim, P. Moskovits, The definitive guide to HTML5 WebSocket, Vol. 1, Springer, 2013. doi:10.1007/978-1-4302-4741-8. URL https://doi.org/10.1007/978-1-4302-4741-8

[63] 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. URL https://doi.org/10.1109/MIPRO.2014.6859715

[64] Site UML specification: https://www.omg.org/spec/UML/About-UML/

[65] Site Astah IDE: http://astah.net/

[66] Grand, M., 2002. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML. (2nd ed.) John Wiley & Sons, Inc. New York

[67] Larman, C., 2004. Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process, Prentice Hall

[68] Site Android studio: https://developer.android.com/studio/