The Version of Record of this manuscript has been published and is available in the International Journal of Human-Computer Interaction at https://dx.doi.org/10.1080/10447318.2022.2098907.

GearWheels: A Software Tool to Support User Experiments on Gesture Input with Wearable Devices

Ovidiu-Andrei Schipor and Radu-Daniel Vatavu

MintViz Lab, MANSiD Research Center, Ştefan cel Mare University of Suceava
13 Universitatii, 720229 Suceava, Romania

ARTICLE HISTORY
Compiled June 29, 2022

ABSTRACT
We introduce GearWheels, a software tool designed to support user studies and experiments on topics about gesture input with wearables, including smartwatches, smart rings, and smartglasses. GearWheels features an event-based asynchronous software architecture design, implemented exclusively with web standards, communications protocols, data formats, and programming technology, which makes it flexible and encompassing of many programmable wearable devices that support HTTP and WebSocket communications via a Wi-Fi connection. GearWheels differentiates from prior software tools for gesture acquisition, elicitation, recognition, and analysis with its web-based, wearable-oriented, and experiment-centered software architecture and user interface design. We demonstrate GearWheels by setting up an experiment with a touchscreen device affixed to the users' index finger, wrist, and temple of a pair of glasses to illustrate touch stroke-gesture and motion-gesture input acquisition in three experimental conditions involving a smart ring, a smartwatch, and a pair of smartglasses, respectively. We also perform a technical evaluation of GearWheels in the form of a simulation experiment, and report the request-response time performance of the software components of GearWheels with two low-end and high-end smartwatch devices, an Augmented Reality smartglasses, and a smartphone used as the control condition. We release GearWheels as open source software in the community to assist researchers and practitioners in implementing user experiments and studies involving gesture input with wearable devices and to foster new developments and discoveries about gesture input with wearables.

KEYWORDS
Gesture input; wearables; user experiments; user studies; software tool; event-based software architecture.

1. Introduction

Gestures represent an expressive means of non-verbal communication to convey intentions and meaning through movements performed with the fingers, hands, head, and the whole body. Consequently, the expressiveness and versatility of gestures have been extensively exploited for designing interactions with computer systems and devices (Chen, Ma, Peng, Zhou, Yao, Ma, Wang, Gao and Shen, 2018; Gheran, Vanderdonckt and Vatavu, 2018; Leiva, Martín-Albo, Plamondon and Vatavu, 2018; Malu, Chundury and Findlater, 2018; Narayana, Beveridge and Draper, 2018; Ungurean,

CONTACT Radu-Daniel Vatavu. Email: vatavu@eed.usv.ro

Pentiuc and Vatavu, 2009; Vatavu, 2012; Vuletic, Duffy, Hay, McTeague, Campbell and Grealy, 2019; Yamada, Kakue, Shimobaba and Ito, 2018). Among these, gesture input is particularly useful for interactions with wearable devices that are worn or affixed to the body and, consequently, have a distinctive vantage point to capture gesture input and provide corresponding feedback to the user. These devices can sense, recognize, and respond to the movements of the body (Bailly, Müller, Rohs, Wigdor and Kratz, 2012; Chen, 2015; Cioată and Vatavu, 2018; Esteves, Velloso, Bulling and Gellersen, 2015; Gheran et al., 2018; Kim, Kwon, Han, Park and Jo, 2018) with a diversity of form factors, functions, and built-in sensors, from computers that can be worn, information appliances that can be worn, and computers integrated into clothing, respectively (Baber, 2001). Wearables also enable gesture-based interactions that leverage the user's body for both input and output, *e.g.*, Pose-IO (Lopes, Ion, Mueller, Hoffmann, Jonell and Baudisch, 2015) accepts user input via gestures performed with the wrist and provides output via electrical muscle stimulation.

Designing effective gesture-based interactions for wearables subsumes a corresponding body of scientific and design knowledge about users' preferences for interactive gestures (Dingler, Rzayev, Shirazi and Henze, 2018; Gheran et al., 2018; Zaiţi, Pentiuc and Vatavu, 2015) and users' abilities to perform gestures (Malu et al., 2018; Oh and Findlater, 2014; Vatavu and Ungurean, 2019). This knowledge can be obtained by employing models of gesture input and by conducting user experiments and studies to collect and analyze new data. While theoretical models are useful to inform the design of gesture sets for interactive computer systems, *e.g.*, estimation of articulation speed from the geometrical shape of the gesture (Leiva, Vatavu, Martin-Albo and Plamondon, 2020), their availability is limited. Moreover, such models (Cao and Zhai, 2007; Leiva, Martín-Albo and Plamondon, 2015; Leiva et al., 2018, 2020; Long, Landay, Rowe and Michiels, 2000; Vatavu, Vogel, Casiez and Grisoni, 2011) haven't been validated for gesture input with wearables, and further examinations are needed to understand their applicability outside the scope and context in which those models were introduced. The alternative is conducting user studies and experiments, which represent an important source of information for researchers and practitioners to understand users' gesture articulation performance, preferences, and behavior during gesture input with various types of wearable devices. However, this approach means having access to specialized software to collect gestures from wearable devices that come with heterogeneous operating systems, hardware specifications, sensor types, sensor resolution, and dependency on software libraries for specific platforms.

In this context, flexible software tools are needed to assist researchers and practitioners with gesture collection during user studies and experiments. Overall, tools are important in HCI to support design and implementation of user interface software (Myers, 1995) and rapid prototyping (Beaudouin-Lafon and Mackay, 2002), and to understand user behavior regarding interactive computer systems (Ali, Morris and Wobbrock, 2019; Vanderdonckt, Zen and Vatavu, 2019; Vatavu and Wobbrock, 2015). Tools, as a specific type of research contribution in HCI (Wobbrock and Kientz, 2016), have been particularly welcomed at HCI venues, *e.g.*, the User Experience and Usability subcommittee of CHI encourages papers that extend the knowledge, practices, methods, components, and tools that make technology more useful, usable, and desirable; also see Terenti and Vatavu (2021) for an overview of the practices adopted by HCI researchers to describe their software tools in academic publications. In this context, several tools have been released in the scientific community of HCI to support gesture collection, elicitation, recognition, and analysis (Bomsdorf, Blum and Künkel, 2017; Kazemitabaar, McPeak, Jiao, He, Outing and Froehlich, 2017; Kin, Hartmann,

DeRose and Agrawala, 2012a; Kin et al., 2012a; Kin, Hartmann, DeRose and Agrawala, 2012b; Long Jr, Landay and Rowe, 1999; Lü, Fogarty and Li, 2014; Lü and Li, 2012; Magrofuoco, Roselli, Vanderdonckt, Pérez-Medina and Vatavu, 2019; Nebeling, Teunissen, Husmann and Norrie, 2014; Vatavu, 2019; Vatavu and Wobbrock, 2015). However, tools for wearable devices have been mostly focused on construction kits (Jones, Nabil, McLeod and Girouard, 2020; Kazemitabaar et al., 2017), leaving researchers and practitioners with little options other than implementing gesture data collection software by themselves for their own gesture input collection studies and experiments. For example, in a study about identifying consistent gestures across smartphones, watches, and glasses, Dingler et al. (2018) "used Apache Cordova 3 to create a corresponding Android app optimized for each device type, namely the Samsung Galaxy S5 smartphone (running Android version 5.0), the Samsung Galaxy Gear SM-V700 smartwatch (running Android version 4.2.2), and Google Glass (running Android version 4.4.4)" (p. 4). When conducting a study about on-body input for mobile users with visual impairments, Oh and Findlater (2014) built a "computer vision module, which tracked the color marker on the participant's gesturing finger, [...] and custom software running on a laptop [...]. The custom touch-detection module ran on an Arduino Leonardo board [...] to detect capacitive input" (p. 117). When describing a study about gestures performed by people with motor impairments using smartphones and tablets, Vatavu and Ungurean (2019) noted: "A custom Android application was developed to implement the experiment design and to collect stroke-gestures" (p. 215:3) and, in a follow-up (Vatavu and Ungurean, 2022) addressing similar gesture types for wearables: "we developed a custom Tizen Web application to collect stroke-gestures with the integrated touchscreen [...] and motion-gestures with the built-in 3-axis accelerometer" (p. 4). In an end-user gesture elicitation study involving mid-air gestures for television, Zaiţi et al. (2015) described their apparatus as follows: "A 40-in. (102 cm) Sony TV and a Leap Motion controller were connected to a computer running Microsoft Windows 8.1 and our custom gesture acquisition software that implemented the experiment design" (p. 823). Many other examples such as these can be encountered in the scientific literature.

In this context, we introduce GearWheels (Gesture Experiments ARchitecture for Wearables with HEterogeneous ELementS), a tool with a dedicated software architecture designed to assist researchers and practitioners in their user studies and experiments involving gesture input with wearable devices. Just like the gearwheels of a machine composed of multiple elements and interconnected parts, we designed the multiple components of the GearWheels software architecture to provide flexible functionality for a variety of wearable devices that support gesture input. GearWheels is flexible and encompassing of many types of wearables, and can be employed in user experiments to collect touch input, multitouch gestures, stroke-gestures, and motion-gestures from programmable wearable devices that support HTTP and WebSocket communications via a Wi-Fi connection. Our practical contributions are as follows:

(1) GearWheels, an event-based asynchronous software architecture design and a corresponding software tool for implementing gesture input experiments with programmable wearables that connect to Wi-Fi and support HTTP and WebSocket communicates. The design of GearWheels was informed by nine quality properties (*e.g.*, modularity, reusability, replaceability, interoperability, etc.) formulated as requirements and implemented via nine software development strategies (*e.g.*, asynchronous processing, web-based communication, inversion of control and dependency injection, etc.). We present the technical details of Gear-

Wheels and a characterization of its software architecture based on the Systems and Software Quality Requirements and Evaluation ISO/IEC standard. Also, we show how GearWheels differentiates from prior work with its unique combination of web-based, wearable-oriented, and experiment-centered functionality.

(2) We demonstrate the practical usage of GearWheels by describing the steps of an experiment involving a small touchscreen device affixed to various parts of the body—on the hand in the form of a ring device, on the wrist as a smartwatch, and at head level, attached to the temple of a pair of glasses—to exemplify stroke-gesture and motion-gesture collection with GearWheels.

(3) We perform a technical evaluation of the GearWheels software architecture in terms of the request-response time of its software components. We conduct the evaluation in the form of a within-subjects simulation experiment involving two smartwatch devices, a pair of smartglasses, and a smartphone used as the control condition.

We release GearWheels as an open-source tool in the community to assist researchers and practitioners in implementing their own user studies and experiments involving gesture input with wearable devices. In doing so, we hope to foster new discoveries and regarding users' gesture articulation performance and gesture preferences during interactions with wearables.

2. Related Work

We discuss in this section systems and tools developed within the HCI community to support implementation of user experiments and studies about gesture input, and tools to assist with gesture data collection and analysis. We focus on the body of research addressing gesture input and, where available, gesture input with wearable devices. We also relate to aspects of software architecture design and, specifically, to asynchronous software architectures due to their capability to handle heterogeneous input and output devices (Schipor, Vatavu and Vanderdonckt, 2019a; Schipor, Vatavu and Wu, 2019b) and, thus, their relevance to our scope.

2.1. Software Architecture and Tools in HCI

We understand by software architecture the abstract, yet implementable, structural and functional description of a software system (Bass, Clements and Kazman, 2003; Clements, Garlan, Little, Nord and Stafford, 2003) that impacts the five stages of a system's life cycle, from requirements specification to development, testing, deployment, and influence on maintenance strategies (Breivold, Crnkovic and Larsson, 2012). Among the many approaches to software architecture design (Aleti, Buhnova, Grunske, Koziolek and Meedeniya, 2012; Banijamali, Pakanen, Kuvaja and Oivo, 2020; Kruchten, Obbink and Stafford, 2006), asynchronous design focuses on units of work that generate and process events that are temporally independent from the main flow of the system (Hauck, 1995; Schmidt, Stal, Rohnert and Buschmann, 2013). Asynchronous processing is especially important in large and complex interactive computer systems, such as smart environments (Schipor et al., 2019a,b; Schipor, Wu, Tsai and Vatavu, 2017), where multiple devices and sensors—from the environment, but also handheld, worn, and operated by users—stream data simultaneously.

One popular strategy to implement asynchronous software applications is event-

driven programming, where the processing flow is dictated by the occurrence of external events, *e.g.*, data from sensors (Schipor et al., 2019b), messages delivered by third-party applications and services, and user input for interactive systems (Schipor et al., 2019b). Event-driven programming enables low coupling among the multiple intercommunicating components of a system. Events are encapsulated in the form of messages, which implement notifications that travel in the software architecture (Chandy, 2006). During the inter-component communications, some of the software components act as *producers* and generate events, while others as *consumers* that receive messages and process the corresponding events (Moxey, Edwards, Etzion, Ibrahim, Iyer, Lalanne, Monze, Peters, Rabinovich, Sharon et al., 2010). Since events reflect changes in the state of the producers, they are strongly coupled with their source. Event-driven software architectures for implementing interactions with computer systems, such as Euphoria (Schipor et al., 2019a) for interactions in smart environments, and SAPIENS (Schipor et al., 2019b) for peripheral interaction, specifically require the implementation of producer and consumer software components according to their operation logic and data flows.

Of specific interest to the scope of our work are systems, tools, platforms, and software architecture designs that accept input from users (*i.e.*, the participants of an experiment or study) and, thus, place users at core of the informational ecosystem (García-Holgado and García-Peñalvo, 2016). From this perspective, the intersection between software engineering and HCI in the scope of our work encompasses "the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them" (Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank, 1992). In this regard, Cruz-Benito *et al.* (Cruz-Benito, Garcia-Penalvo and Theron, 2019) identified four types of architectural solutions for HCI applications: layered, module and component based, web services, and agent based. Jones, Milic-frayling, Rodden and Blackwell (2007) proposed a contextual method for improving software products based on the discovery and evaluation of new features for standard web clients. Also, the data traveling through the software architecture can take many forms according to the available input modalities, from mid-air gestures (Vatavu, 2017c) to voice input (Aiordăchioae, Schipor and Vatavu, 2020), touch-based interactions (Vatavu, Gheran and Schipor, 2018), eye gaze input (Gherman, Schipor and Gheran, 2018), electroencephalography signals (Schipor, Pentiuc and Schipor, 2012), and other physiological measurements (Schipor, Pentiuc and Schipor, 2011). Therefore, a strong relationship can be identified between the software architecture of an interactive computer system and its interaction capabilities, which can be characterized with quality design requirements. These include support for multimodal interaction (Gherman et al., 2018), context awareness (Roda, Navarro, Zdun, López-Jaquero and Simhandl, 2018; Schipor et al., 2019b), enhanced usability (Aiordăchioae et al., 2020; Vatavu et al., 2018), support for collaborative work (Tănase, Vatavu, Pentiuc and Graur, 2008), reusability and replaceability (Schipor, Bilius and Vatavu, 2022), and others. The software architecture of GearWheels, the tool that we introduce in this work for gesture input experiments with wearable devices, was informed by nine quality requirements; see Subsection 3.1.

2.2. Software Tools for Gesture Experiments and Studies

Several tools exist to assist researchers and practitioners interested in conducting experiments and studies about gesture input and to analyze data from such studies. To

identify prior work, we used surveys of the literature on gesture-based input (Magrofuoco, Roselli and Vanderdonckt, 2021; Siean and Vatavu, 2021; Villarreal-Narvaez, Vanderdonckt, Vatavu and Wobbrock, 2020; Vuletic et al., 2019), and performed focused literature searches[1] using the ACM DL and IEEE Xplore electronic databases. We found tools and systems designed for gesture acquisition (Acharya, Matovu, Serwadda and Griswold-Steiner, 2019; Magrofuoco et al., 2019), gesture processing (Bomsdorf et al., 2017; Kazemitabaar et al., 2017; Nebeling et al., 2014), sharing gesture sets via online repositories (Magrofuoco et al., 2019; Magrofuoco and Vanderdonckt, 2019; Solis, Pakbin, Akbari, Mortazavi and Jafari, 2019), and gesture analysis (Kin et al., 2012a,b; Lü et al., 2014; Vatavu, 2017a, 2019; Vatavu and Wobbrock, 2015), respectively. The gesture types addressed in this prior work can be collected from a variety of input devices, from touchscreens (Kin et al., 2012a; Long Jr et al., 1999; Lü and Li, 2012) to video cameras and depth sensors (Ashbrook and Starner, 2010; Buruk and Özcan, 2017; Magrofuoco et al., 2019; Nebeling, Ott and Norrie, 2015; Nebeling et al., 2014), motion sensors (Tang and Igarashi, 2013), IoT devices (Solis et al., 2019), mobile devices (Acharya et al., 2019; Kohlsdorf, Starner and Ashbrook, 2011; Magrofuoco and Vanderdonckt, 2019), dual-screen mobile devices (Wu and Yang, 2020), and wearables (Jones et al., 2020; Kazemitabaar et al., 2017; Roggen, 2020). Some of these systems implement distributed software architectures, *e.g.*, on the web and in the cloud, to enable remote access to the tool, data, and resources (Ali et al., 2019; Buruk and Özcan, 2017; Magrofuoco and Vanderdonckt, 2019; Schipor et al., 2019a; Solis et al., 2019). We found that wearable-oriented tools leveraged a variety of sensors to offer insights about users' everyday activity, but also to provide feedback. Some of the systems specifically designed for wearables are 360QS (Singh, Fernandez-Luque and Srivastava, 2017), a toolkit for research on sleep and physical activities, SenseGraph (Alpers and Benţa, 2021), an application for the assessment of affective states, Compressables (Endow, Moradi, Srivastava, Noya and Torres, 2021), a prototyping toolkit for compression-based haptic feedback, and WDK (Haladjian, 2019), a development kit for applications that employ activity recognition.

By surveying the scientific literature, we identified four types of tools designed for gesture input: (1) tools for gesture set design represented by software applications that enable users to manage gesture sets for interactive systems (Ashbrook and Starner, 2010; Kin et al., 2012b; Kohlsdorf et al., 2011; Long Jr et al., 1999; Solis et al., 2019; Tang and Igarashi, 2013), (2) gesture acquisition tools represented by software applications that collect and store gestures (Acharya et al., 2019; Magrofuoco et al., 2019; Magrofuoco and Vanderdonckt, 2019), (3) gesture recognition tools with services and features for gesture processing and classification (Kin et al., 2012a,b; Lü et al., 2014; Lü and Li, 2012), and (4) experiment-centered tools represented by software applications designed to assist researchers and practitioners in conducting user studies and experiments about gesture input (Ali et al., 2019; Buruk and Özcan, 2017; Magrofuoco and Vanderdonckt, 2019; Nebeling et al., 2015). For example, CUBOD (Tang and Igarashi, 2013) enables definition of custom body gestures and MAGIC (Ashbrook and Starner, 2010; Kohlsdorf et al., 2011) assists designers in creating gesture sets that minimize false positives for specific gesture recognition approaches. GestMan (Magrofuoco et al., 2019) is a cloud-based tool for the management of gesture sets that enables researchers and practitioners to remotely collect gestures from end users via HTML web appli-

[1]We ran the query https://dl.acm.org/action/doSearch?fillQuickSearch=false&target=advanced&expand=dl&AllField=Title%3A%28wearable*+AND+tool*%29 in the ACM DL and https://ieeexplore.ieee.org/search/searchresult.jsp?action=search&matchBoolean=true&newsearch=true&queryText=(%22Document%20Title%22:wearable*%20AND%20%22Document%20Title%22:tool*) in IEEE Xplore.

Table 1. Comparison of prior work using a set of criteria representing features relevant to GearWheels.

	Wearable orientation	Web based	Open source	Experiment centered
360QS (Singh et al., 2017)	●	●	◔	◑
Compressables (Endow et al., 2021)	●	●	●	◔
Crowdlicit (Ali et al., 2019)	○	●	●	●
Cubod (Tang and Igarashi, 2013)	◔	○	◔	◑
Gelicit (Magrofuoco and Vanderdonckt, 2019)	○	●	●	●
GestAnalytics (Buruk and Özcan, 2017)	○	○	●	◑
GestMan (Magrofuoco et al., 2019)	◔	●	●	○
GestureCoder (Lü and Li, 2012)	◔	○	○	◑
GestureScript (Lü et al., 2014)	◔	○	●	◑
GestureAnalyzer (Jang, Elmqvist and Ramani, 2014)	○	○	◔	●
KinectAnalysis (Nebeling et al., 2015)	◔	●	●	◑
Magic (Ashbrook and Starner, 2010)	◑	○	○	◑
Magic 2.0 (Kohlsdorf et al., 2011)	◑	●	○	◑
Proton (Kin et al., 2012b)	◑	○	◔	◔
Proton++ (Kin et al., 2012a)	◑	○	◑	◑
SenseGraph (Alpers and Benţa, 2021)	●	◑	◔	◔
WDK (Haladjian, 2019)	●	◑	●	◑
XDKinect (Nebeling et al., 2014)	◑	●	◑	◔
Overall	◑	◑	◑	◑

cations. Gesture Script (Lü et al., 2014) and Gesture Coder (Lü and Li, 2012) were designed to assist developers in creating gesture classifiers with example-based learning, while multitouch gestures are represented as regular expressions in Proton (Kin et al., 2012a,b). Kinect Analysis (Nebeling et al., 2015), designed for whole-body gestures, features recording, playing, and analysis of gestures, and can be accessed from remote devices via the web. Crowdlicit (Ali et al., 2019) and Gelicit (Magrofuoco and Vanderdonckt, 2019) enable researchers to conduct elicitation studies (Vatavu and Wobbrock, 2015; Wobbrock, Morris and Wilson, 2009) outside the lab and via the web to increase accessibility and replicability (Gheran, Villarreal-Narvaez, Vatavu and Vanderdonckt, 2022) and save resources (Reinecke and Gajos, 2015).

2.3. Summary

The HCI community has put special emphasis on empowering researchers and practitioners with tools to support the design and implementation of user experiments and studies, and our literature review identified several such tools for gesture acquisition, elicitation, recognition, storage and management, and analysis. These tools have been implemented using a variety of software architecture styles and design principles, from desktop-based to SaaS and PaaS. Among these approaches, we highlight the advantages of event-based, asynchronous processing software and of applications that are readily available over the web that use web standards and data formats. However, wearable devices, such as smartwatches and smartglasses, have been little addressed by such tools. There is thus the need to complete the landscape of gesture toolkits, platforms, frameworks, and analysis software with tools that specifically address wearables and solutions that handle appropriately their heterogeneous characteristics. To this end, we present in Section 3 GearWheels, our software tool for studies and experiments involving gesture input with wearable devices. In Table 1, we provide an

overview of the prior work surveyed in this section, characterized according to four features representative of GearWheels: wearable orientation, web-based and open source development, and experiment-centered design, respectively.

3. GearWheels

We present the software architecture details of GearWheels in relation to a set of nine design requirements that we outlined for the application context of providing support for gesture input studies and experiments with wearable devices. Just like the gearwheels that work together to provide the functionality of a complex machine, GearWheels consists of several intercommunicating and interoperating software components that are composed of software modules dedicated to specific tasks in the architecture. The components are those parts of the software that run independently, although they cooperate and exchange information, while the modules are implementing specific tasks, services, and functionality. We start our description of GearWheels with a presentation of its design requirements.

3.1. Design Requirements for GearWheels

Designing software architecture is a process that often starts with the identification of the quality attributes that the final system must meet (Evesti, 2007). Some approaches consider quality attributes subsumed to concrete usage scenarios (Folmer, van Gurp and Bosch, 2003), such as the Software Architecture Analysis Method (Kazman, Bass, Abowd and Webb, 1994), or to external constraints, such as the Architecture-Based Component composition/Decision-oriented Design (Cui, Sun, Xiao and Mei, 2009). In this work, we employ the Software Product Quality Model from SQuaRE, the Systems and Software Quality Requirements and Evaluation ISO/IEC standard (International Organization for Standarization, 2005), from which we select nine quality properties directly relevant for GearWheels; see Figure 1 for a visual illustration. In the following, we group these properties into three categories according to the interactions that they enable among software modules, components, and between users and GearWheels.

The first category of quality properties regards the interaction among the software modules located within the same component, as follows:

- R_1. **Modularity** represents the degree in which a software component can be decomposed into low-coupled software modules so that any modification within the individual modules propagates little or no changes in the other modules. This property is key for software tools designed to support and conduct user experiments, because such tools need to accommodate various experiment designs, including designs that were not envisaged or available when the software components and modules were originally developed and integrated to form the tool. Consequently, the software architecture of GearWheels needs to isolate potential changes within the various software modules and components.
- R_2. **Reusability** measures the ability of a software module to integrate more than one software component without modification. This requirement is useful when specifying user experiments in order to easily activate and deactivate various functionalities of the tool.
- R_3. **Testability** refers to the degree in which a software module can be independently tested without the need to recreate its entire software ecosystem. This property

implies the existence of rigorously defined and strongly separated responsibilities for each software module. For the success of user experiments conducted with GearWheels, failures should not occur with the software modules, components and, overall, the tool while running the experiment. For instance, automatic testing is a desirable feature to ensure that data represented by user input is recorded and stored correctly by the tool.

Our second category of requirements refers to the interaction between the software components of the GearWheels architecture. Since components have a higher level of independence compared to modules, they should interact in ways that are more loosely coupled. Our design requirements for software modules apply to software components as well, but the latter also need to comply with additional requirements, as follows:

R_4. **Co-existence** represents the degree to which a software component shares the same software ecosystem with other components without producing any harmful influence on their functionality and/or efficiency. For instance, it is common in user experiments with interactive computer systems for participants to interact with multiple devices, *e.g.*, in the context of cross-device input (Brudy, Holz, Rädle, Wu, Houben, Klokmose and Marquardt, 2019) or for applications running in smart environments (Schipor et al., 2019a,b).

R_5. **Interoperability** is the ability of two or more software components to exchange data via a common standard and to use that data during their operation. This property is relevant to GearWheels since heterogeneous devices are often considered in user experiments to collect diverse information about participants' input behavior, *e.g.*, an eye-tracking device, touchscreen, and EEG headset were used in conjunction by Gherman et al. (2018) to understand how users with motor impairments interact via touch input, and Dingler et al. (2018) examined consistent gestures across smartphones, watches, and glasses.

R_6. **Replaceability** refers to the degree in which a software component can be replaced by another without generating a significant impact on the overall functioning of the architecture. Moreover, the replacement must remain transparent for the users of the tool implementing the architecture, given that the functionality of the software architecture is preserved. This property plays an important role when implementing multiple experimental conditions in a user study, *e.g.*, by considering different mobile devices during an experiment designed to evaluate children's performance with touch input (Vatavu, Cramariuc and Schipor, 2015), or when conducting the study in multiple locations (Vatavu and Ungurean, 2019) or using participants' own devices (Buzzi, Buzzi, Leporini and Trujillo, 2017).

We also identified three design requirements that regard the quality of the interaction between GearWheels and its users, either the experimenter that sets up the study or the participants that follow the instructions provided by GearWheels:

R_7. **Appropriateness** represents the degree in which the functionality of GearWheels is in line with the expected tasks and goals of the experiment. This requirement can be extended to system completeness, a quality property that indicates whether the existing functionality covers all of the objectives that were specified for the system.

R_8. **Operability** denotes how easy it is for users to configure, control, and operate GearWheels. For this requirement, we identify two user roles: the experimenter that sets up the study and runs it and the participants of the study that receive instructions from GearWheels during the experiment.

R_9. **Learnability** specifies the degree in which GearWheels helps its users to operate

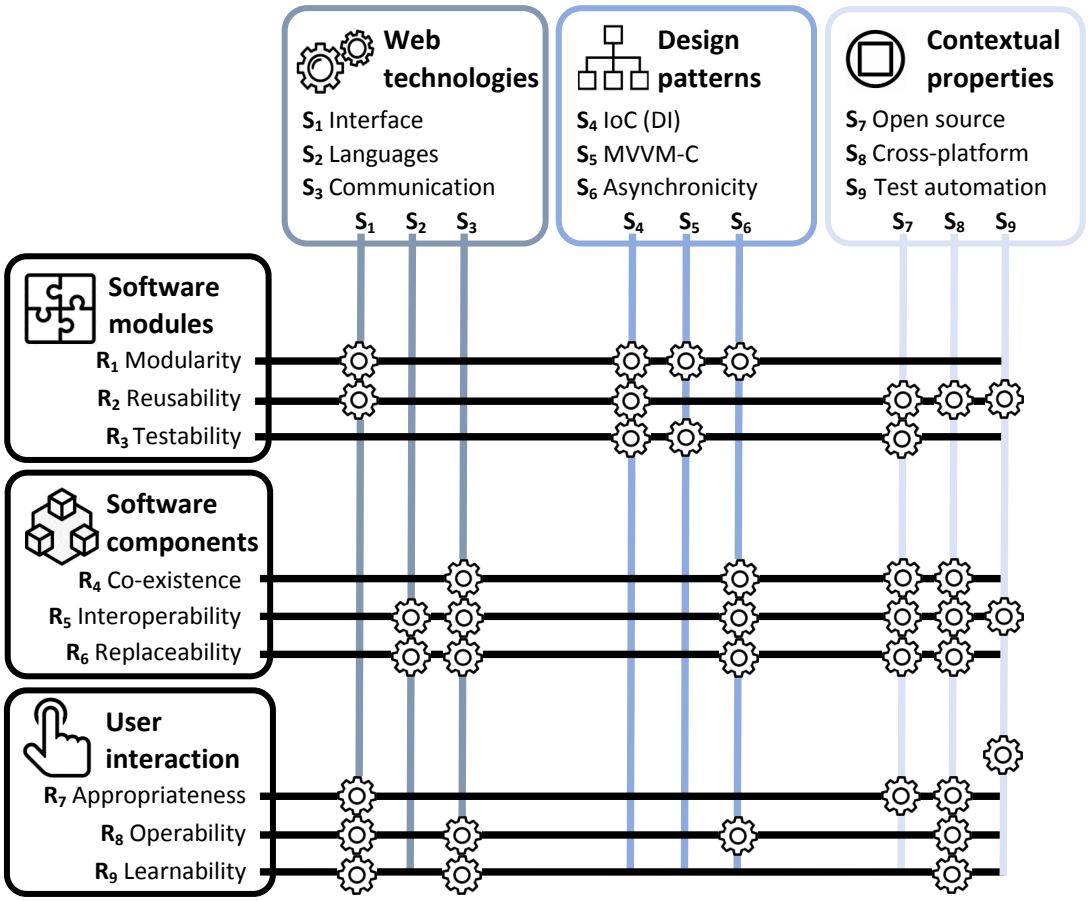

Figure 1. Visual overview of the nine quality properties, represented using requirements (left) and strategies (right), for the software architecture design of GearWheels. Relationships between requirements and strategies are highlighted, *i.e.*, each requirement is met via several strategies, and each strategy is involved in addressing several requirements.

it effectively during the experiment. Although the study participants can ask the experimenter for assistance, high learnability is always preferable to prevent unnecessary delays caused by misunderstandings or repetitions of trials.

3.2. Strategies for Implementing the Design Requirements of GearWheels

We elaborated nine strategies, grouped into three categories—technology-related, design patterns, and contextual—to implement the design requirements set for GearWheels; see Figure 1 for a visual illustration. Our first category of strategies addresses technology to implement the GearWheels software architecture and tool. We rely on software technology for the web since it is supported by many wearable devices with a web browser and Wi-Fi connection, and is mature enough for our purposes to enable access to device resources through standardized APIs, such as the accelerometer from a smartwatch, touch events for touchscreen displays, retrieving frames from the built-in video camera and microphone of a smartglasses device, etc.

S_1. **Web interface** refers to the use of HTML elements in the presentation layer of GearWheels. This strategy implements the modularity and reusability require-

ments (R_1 and R_2 in Figure 1) since several HTML elements can form web components as atomic, easily reusable pieces of software; see Ast and Gaedke (2017). Also, web applications should be familiar to the study participants and, thus, previously acquired interaction experience with computing systems is reused toward effective interaction with GearWheels (requirements R_7 to R_9).

S_2. **Web languages** are represented by HTML, CSS, and JavaScript used for the specification of the structure, presentation, and behavior layers of GearWheels. This strategy enables portability since the majority of smart devices feature web browsers or can be programmed using languages for the web.[2] Moreover, JavaScript can also be used as a server-side language on various platforms, powered for instance by the node.js runtime environment.[3]

S_3. The **web communication** strategy specifies that all the communications between the software components of GearWheels are implemented via web protocols, such as HTTP for half-duplex client-server communications and WebSocket for full-duplex communications. This strategy offers standardized interaction between all the software components of GearWheels (requirements R_4 to R_6).

Our second category of strategies encompasses three software design patterns that describe how software objects are created, organized, and instantiated, as follows:

S_4. **IoC-DI** refers to the inversion of control (IoC) through dependency injection (DI). In this approach, dependencies are injected into the object rather than created inside the object. This pattern assures a clear separation between the creation of a software object and its use as well as the decoupling between high and low level classes and, thus, assures optimized interaction among the software modules of the same component (design requirements R_1 to R_3).

S_5. **MVVM-C** is an approach derived from the Model, View, and Controller (MVC) approach to software design (Arcos-Medina, Menéndez and Vallejo, 2018) that groups the software objects within a module in four categories: models that embed domain data along the business logic, views for the user interface, viewmodels that abstract views, and the coordinator for handling multiple viewmodels. The main benefits of this structural design pattern are the decoupling of objects (design requirement R_1) and simple testing (design requirement R_3).

S_6. **Asynchronous flow** specifies that GearWheels handles asynchronous interactions among its software components (design requirements R_4 to R_6). This aspect is important since multiple input devices may be used at the same time during an experiment (requirement R_8). GearWheels must handle unpredictable time-response behaviors and integrate them accordingly in the general flow of the experiment. This approach relies on JavaScript that offers mechanisms to handle asynchronicity, such as callbacks, promises, and async/await code execution patterns; see Schipor et al. (2019a,b) for examples of event-based software architectures implemented using JavaScript exclusively.

Our last category of strategies to implement the design requirements of GearWheels groups three contextual strategies with potential impact on the community of researchers and practitioners that wish to conduct user experiments and studies involving gesture input. By adopting these strategies, GearWheels is free to use, but also open to be extended and, thus, to meet other requirements as needed in the future.

S_7. **Open source** indicates our reliance on open-source technologies, but also that

[2]See, for example, Tizen Web Application development, consisting of HTML, JavaScript, and CSS combined in a package that can be installed on a Tizen device, https://docs.tizen.org/application/web/index

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

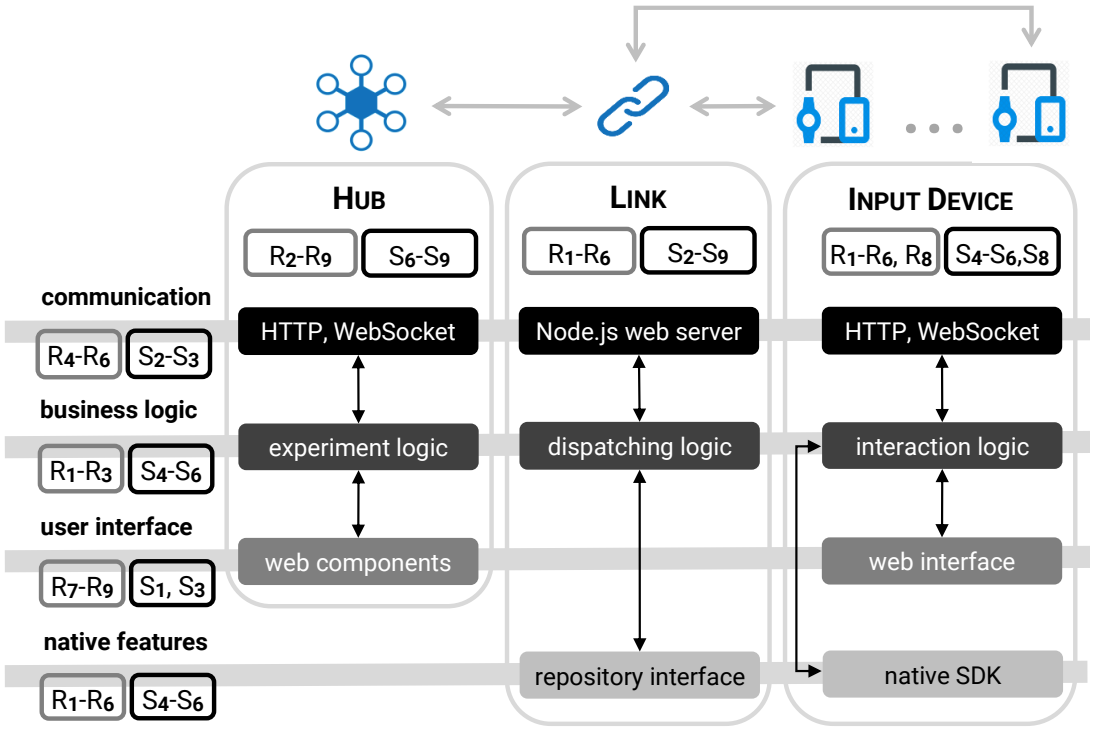

Figure 2. Overview of the GearWheels software architecture highlighting components (top), layers (left), and supporting technologies (center) to implement them.

GearWheels is freely available under an open-source license. Our choice to use open-source technology has a positive impact on both the interaction among software modules (requirements R_2 and R_3) and software components (R_4 to R_6), because third parties can reuse and test parts of GearWheels as needed.

S_8. **Cross-platform** refers to the possibility to run GearWheels on platforms with various hardware and software configurations. This property is important because of the variety of heterogeneous input devices that may be employed during user experiments with gesture input; see our corresponding discussion from the requirements specification section. Cross-platform compatibility is achieved via web technologies, such as web data formats and communications protocols.

S_9. **Test automation** specifies that GearWheels supports testing via simulated data (design requirement R_3). The testing files specify system behavior in various conditions (requirement R_7) and facilitate the refactoring of the software (R_6) since the tests can be automated.

3.3. Software Architecture

We implemented the software architecture of GearWheels starting from our nine design requirements and using the proposed strategies to address those requirements. GearWheels consists of three software components, InputDevice, Hub, and Link, illustrated in Figure 2:

- The InputDevice software component is designed to run on a wearable device, such as smartwatch, smartglasses, etc., and receives direct input from the user, *e.g.*, a tap on the touchpad embedded in the temple of the smartglasses. To

maximize portability (according to design requirements R_4 to R_6), we implemented the InputDevice component using web technologies only (strategies S_2 to S_3). Although JavaScript offers mature support to handle a variety of user and system events, such as touch input and video camera data, adapter modules are necessary when external sensors, such as eye trackers, are used and that are interfaced via dedicated SDKs (strategy S_8), *e.g.*, for eye gesture input with smartwatches (Esteves et al., 2015). In its current implementation, GearWheels offers support for the acquisition of touchscreen stroke-gestures and accelerometer and gyroscope motion-gestures implemented with JavaScript code that can be readily used as is or adapted to implement specific gesture acquisition scenarios for custom user studies and experiments. A touchscreen stroke-gesture is represented as a series of 2D points with timestamp and stroke ID associated information. A motion-gesture is represented as a series of 6D points representing linear acceleration and orientation data captured on three axes of movement. These two types of gestures are among the most commonly encountered for wearable input and, thus, are supported by GearWheels out of the box with JavaScript API. In Section 4, we present examples of these gesture types collected in various experimental conditions, JSON data formats used in GearWheels to represent gestures, and the JavaScript code available from GearWheels to collect stroke-gestures and motion-gestures.

- The Hub implements the workflow of the user experiment and runs the user interface. It enables the experimenter to navigate through the experiment trials and enter notes during the experiment, *e.g.*, whether a specific input was incorrect (requirements R_7 to R_9). The Hub is logically connected with as many InputDevice components as necessary to implement the experiment design (requirements R_4 to R_6). However, these connections are not point to point, but mediated by the Link component, described next (strategies S_6 and S_8). The Hub is implemented as a responsive web application (strategies S_1 to S_3) to maximize its capacity to run on many platforms. For GearWheels to function as an integrated system (requirements R_1 to R_3), the Hub and the InputDevice software components communicate via the Link; see Figure 2.
- The Link software component is a node.js application (strategies S_2 and S_3) that mediates the asynchronous communications between all of the other components (S_3 and S_6). It establishes full-duplex WebSocket connections and dispatches messages between the Hub and various instances of InputDevice corresponding to each of the wearables used in the gesture input experiment. Link also hosts a HTTP server that runs the Hub web application and user interface. By adopting this design approach, the Hub can save the data collected during the experiment from the various instances of InputDevice directly to the file system of the server. Although the data flow includes the Link, the business logic is mainly located within the Hub (requirements R_1 to R_3).

GearWheels implements a layered architecture to optimize the interactions among its software modules and components and between users and the tool (requirements R_1 to R_9), *e.g.*, the UI layer handles the interaction among GearWheels, the experimenter, and the participants of the study. The Hub controls the flow of the experiment and presents the details of the current trial to the participant. Each InputDevice hosts the interaction logic, such as the knowledge of how to process the multiple strokes of a gesture produced on the touchscreen display of a smartwatch, or how to react to fixations during eye tracking gesture input. The Link dispatches the messages trans-

mitted between the HUB and the various INPUTDEVICE components with an ad hoc addressing system that uses the IDs of those components, *e.g.*, MAC addresses for the wearable devices. The communication layer is built entirely on top of HTTP and WebSocket standards. Each INPUTDEVICE component enables access to the native features layer when user input cannot be collected using just conventional software technology for the web. This layer is highly specialized for the use case scenario addressed by the experiment and the host operating system available on the wearable device, and may reference specific SDKs and libraries. Thus, it is recommended to keep this layer as thin as possible. All the data produced during the experiment are saved by means of the LINK through the repository interface.

4. Use Case Example for GearWheels

We describe in this section the practical aspects of using GearWheels during an actual experiment involving gesture input with wearable devices. Specifically, we set up an experiment in which participants were asked to perform stroke-gestures and motion-gestures using a small touchscreen[4] with a built-in accelerometer and gyroscope worn in the form of a ring, as a watch, and attached to the temple of a pair of glasses, respectively; see Figure 3, bottom. Two steps must be performed by the experimenter: (1) a JavaScript application is implemented for each wearable device to collect touch, multitouch, stroke-gesture, or motion-gesture input. Our tool already provides JavaScript code to capture such gesture types, which can be readily reused either in the form of a web application (*e.g.*, for Tizen devices) or directly in a web page (for devices featuring a web browser); (2) each wearable to be used during the experiment is connected to the LINK component and assigned to an experimental condition by employing the user interface rendered by the HUB software component of GearWheels.

The HUB renders the user interface that assists both the experimenter during setup and the participants during the actual experiment implemented with GearWheels. First, the HUB requires a full-duplex connection with the LINK. Then, each wearable device is connected to the HUB and encoded as a distinct condition of the experiment; see Figure 3, top for a screenshot. Each INPUTDEVICE component must be connected to the LINK in order to be discovered and integrated by the HUB. Once the experiment starts, randomized trials are presented via the user interface to guide the participant during the experiment; see Figure 4. Practical information, such as the number of strokes of a stroke-gesture or the acceleration data of a motion-gesture, can be displayed in the user interface for the experimenter to monitor the experiment trials. The HUB user interface also informs about the status of the other software components, such as the connectivity with the various instances of INPUTDEVICE and the LINK.

The LINK component mediates the communication among the other software components and offers access to the gesture data repository. In our scenario, LINK records stroke-gestures as time-ordered series of 2D points representing the actual touch points of the user's finger on the touchscreen display, and motion-gestures as series of 6D points representing acceleration and rotation data measured along three axes each. Figure 5 shows two JSON-formatted objects representing the data captured for these two gesture types. Besides the actual gesture data, *e.g.*, the acceleration points for the motion-gesture, each JSON object stores information about the device, trial, experi-

[4]The display of a Samsung Gear Fit 2 smartwatch (https://www.samsung.com/us/mobile/wearables/smart-fitness-bands/galaxy-fit2-black-sm-r220nzkaxar/#specs), which we detached from its strap to implement the finger and glasses conditions. The 1.1-inch display has a 126×294 pixel resolution.

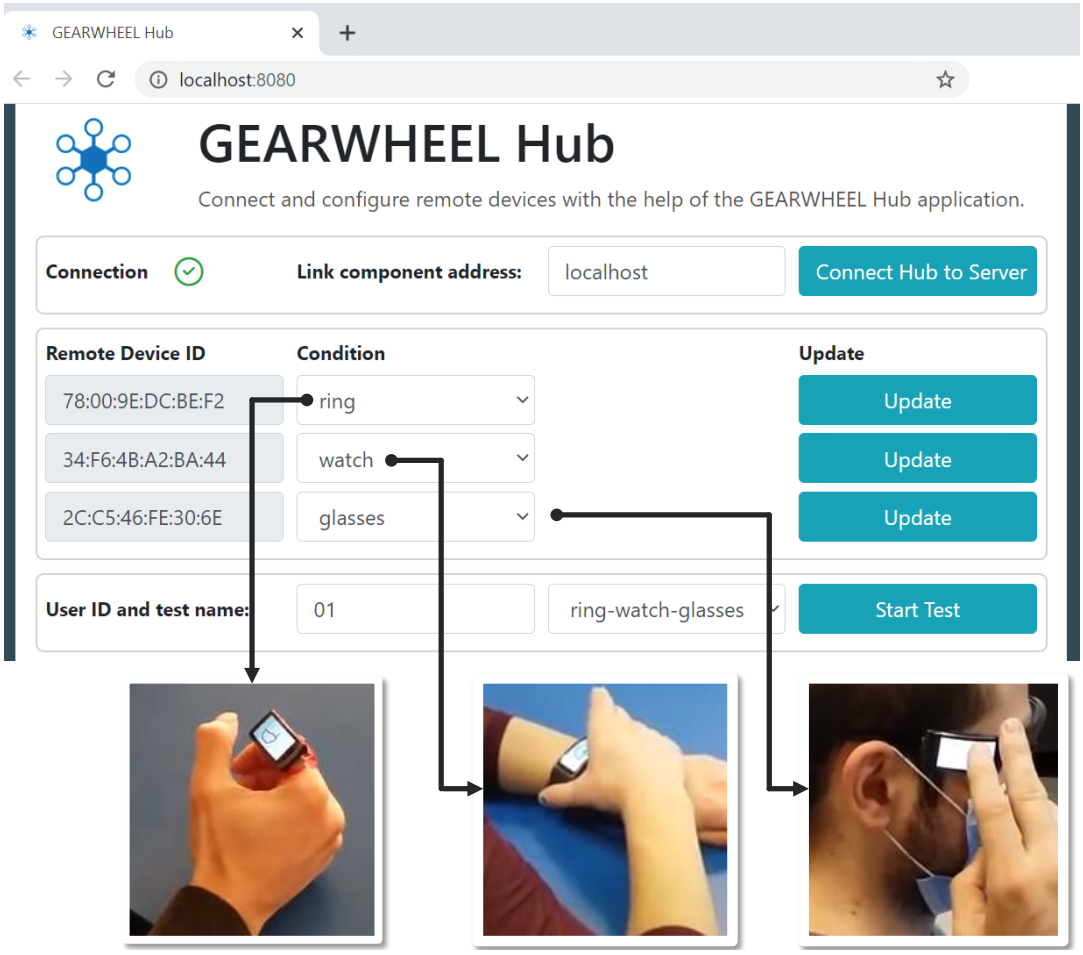

Figure 3. The user interface of the HUB software component specifies the configuration of the experiment with the touchscreen device positioned on the finger (the "ring" experimental condition), on the wrist (the "watch" condition), and attached to the temple of a pair of glasses (the "glasses" condition in this figure).

ment, and the user performing the trial. Figure 6 presents an excerpt of the JavaScript code available in GearWheels to be reused for various INPUTDEVICE components. This code captures stroke-gesture input represented as a series of 2D points by implementing standardized touch up, touch down, and touch move events in HTML and JavaScript. Next, we present three variations of our experiment design, and we illustrate the gesture data collected from several participants.

4.1. Swipe Input on Ring vs. Glasses

In this variation of our experiment, we focused on swipe input represented by directional strokes to the left and right, for which we considered two conditions for wearable devices: (a) swipe input with the thumb performed on a small touchscreen affixed to the index finger and (b) swipe input with the index finger on the temple of a pair of glasses; see Figure 3, bottom-left and bottom-right illustrations. Swipe gestures represent a popular input technique to interact with content on touch displays and devices, from smartphones to smartwatches (Kubo, Shizuki and Tanaka, 2016), rings (Boldu, Dancu, Matthies, Cascón, Ransir and Nanayakkara, 2018), smartglasses (Grossman,

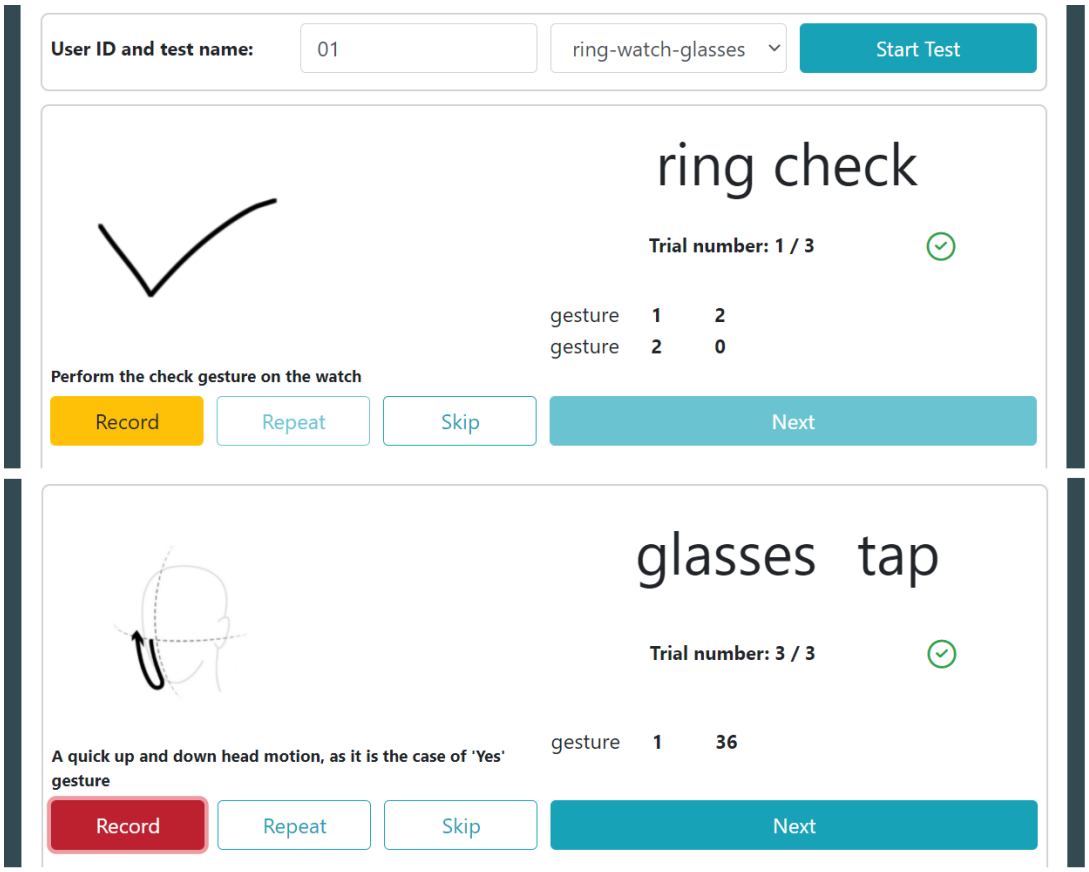

Figure 4. Two trials displayed in GearWheels during the experiment. The first trial (top screenshot) asks the participant to perform a stroke-gesture in the finger-augmentation/ring condition, while the second (bottom screenshot) specifies a gesture performed with rotational movements of the head.

Chen and Fitzmaurice, 2015), and touch-sensitive fabrics (Heller, Ivanov, Wacharamanotham and Borchers, 2014). We used GearWheels to set up the two conditions (ring vs. glasses) with randomized trials and a number of eight repetitions required for each directional swipe. Figure 7 presents the results for each experimental condition and all the participants. At a first glance, there seems to be more variability in the articulation of swipe gestures performed on glasses than on the ring, which could be further verified using relative measures of gesture accuracy (Vatavu, Anthony and Wobbrock, 2013) to draw implications for gesture user interfaces for these devices. However, it is not the goal of this paper to continue with the investigation of users' articulation characteristics of swipe gestures, but interesting questions to examine at this point could be: Are swipes performed with the thumb on a ring worn on the index finger faster and easier to articulate (Vatavu et al., 2011) than swipes performed on the glasses temple? Do users exhibit the same level of articulation accuracy from the straight line of a directional swipe (Vatavu et al., 2013) when performing swipes on the finger compared to the glasses temple? Do users prefer to interact with content displayed on see-through smartglasses using the touchpad embedded in the temple or rather via thumb swipes on the index finger? For which categories of users (Malu and Findlater, 2015) are swipe gestures more convenient to be performed on a touchscreen affixed to the finger than on the glasses?

```
▼ object {11}
    userId : 01
    testId : motion-ring-training
    testType : motion
    device : ring
    deviceId : 78:00:9E:DC:BE:F2
    gesture : pinch thumb
    trialId : 2
    trialStartTime : 1611833239297
    trialEndTime : 1611833246751
    skip : false
  ▼ gestures [1]
    ▼ 0 {2}
      ▼ motionPoints [38]
        ▼ 0 {7}
            aX : -0.072
            aY : 0.002
            aZ : 0.055
            beta : 2.450
            gamma : -0.700
            alpha : -0.280
            T : 1325473810851
        ▶ 1 {7}
          • • •
        ▶ 37 {7}
        numberOfMotionPoints : 38
```

```
▼ object {11}
    userId : 01
    testId : touch-glasses-training
    testType : touch
    device : glasses
    deviceId : 78:00:9E:DC:BE:F2
    gesture : six-point-star
    trialId : 5
    trialStartTime : 1611834145977
    trialEndTime : 1611834155789
    skip : false
  ▼ gestures [1]
    ▼ 0 {2}
      ▼ strokes [2]
        ▼ 0 [167]
          ▼ 0 {4}
              X : 163
              Y : 422
              ID : 0
              T : 1325474716230
          ▶ 1 {4}
            • • •
          ▶ 166 {4}
        ▼ 1 [155]
          ▶ 0 {4}
            • • •
        numberOfStrokes : 2
```


Figure 5. Examples of JSON data for a motion-gesture (left) and a stroke-gesture (right) collected during our test experiment.

4.2. Stroke-Gesture Input on Watches vs. Glasses

In another variation of our experiment, we focused on stroke-gesture input represented by letters, for which we considered two experimental conditions: (a) stroke-gestures articulated on a watch and (b) on the temple of a pair of glasses; see the two conditions illustrated in Figure 3, bottom middle and right, respectively.

Stroke-gestures are convenient as shortcuts (Poppinga, Sahami Shirazi, Henze, Heuten and Boll, 2014), outperform keyboard shortcuts in terms of learning and recall (Appert and Zhai, 2009) and, combined with marking menus (Roy, Malacria, Guiard, Lecolinet and Eagan, 2013), render large sets of commands feasible and easily discoverable on touchscreen user interfaces. We used GearWheels to set up the two experimental conditions (watch vs. glasses) with randomized trials and eight repetitions required for each of the following letters: "X," "M," and "A." (These letters were randomly selected just for demonstration purposes, but they could be associated to actual system functions in a practical application, such as "close," "open menu," and "add new," for instance; see the Augmented Letters technique of Roy et al. (2013) for an

```
function onGestureCompleted() {
    wsClient.send(JSON.stringify({
        deviceId: deviceId,
        operation: "stroke",
        stroke: _points
    }));
    _strokeID = 0;
}

function onTouchDown(x, y, button) {
    document.onselectstart = function() {
        return false;
    }
    document.onmousedown = function() {
        return false;
    }
    if (button <= 1) {
        _isDown = true;
        clearTimeout(_gestureClock);
        x -= _rc.x - getScrollX();
        y -= _rc.y - getScrollY();
        if (_strokeID == 0) {
            _points.length = 0;
        }
        let t = getNowTime();
        _points[_points.length] =
            new Point(x, y, ++_strokeID, t);
    }
}
```

```
function onTouchMove(x, y, button) {
    if (_isDown) {
        x -= _rc.x - getScrollX();
        y -= _rc.y - getScrollY();
        let t = getNowTime();
        _points[_points.length] =
            new Point(x, y, _strokeID, t);
        drawConnectedPoint(
            _points.length - 2,
            _points.length - 1);
    }
}

function onTouchUp(x, y, button) {
    document.onselectstart = function() {
        return true;
    }
    document.onmousedown = function() {
        return true;
    }
    if (button <= 1) {
        if (_isDown) {
            _isDown = false;
            onGestureFinished();
        }
    }
}
```

Figure 6. Example of JavaScript code for recording stroke-gestures with the InputDevice implementation from our experiment involving a touchscreen device.

example.) Figure 8 presents the results for each experimental condition and all the participants. Stroke-gestures performed on the watch visually present higher articulation consistency compared to those performed in the eyes-free context on the glasses, an observation that would be interesting to examine further. However, just like in the case of the previous experiment, it is not the goal of this work to analyze user gesture input, but rather to demonstrate the usage and usefulness of our tool. Investigation options at this point could be: consistency analysis (Anthony, Vatavu and Wobbrock, 2013), comparing stroke-gesture articulations to canonical templates (Vatavu et al., 2013), and evaluating the classification accuracy of popular recognition techniques (Vatavu, Anthony and Wobbrock, 2012a; Wobbrock, Wilson and Li, 2007) for stroke-gestures performed on watches and glasses. These analyzes will reveal useful information about user and system performance regarding letter stroke-gestures articulated on wearables.

4.3. Head Gesture Input for Glasses

In the last variation of our experiment, we addressed gestures performed with movements of the head, for which we considered one condition represented by our touchscreen device attached to the temple of a pair of glasses to measure the linear acceleration of the user's head movements. Head gestures are useful to interact with content displayed on mobile (Hueber, Cherek, Wacker, Borchers and Voelker, 2020) and HMDs (Yan, Yu, Yi and Shi, 2018), to perform target selection in VR and AR (Yan, Shi, Yu and Shi, 2020), and to play video games (Ungurean et al., 2009). We used GearWheels to set up an experiment with one wearable device (the glasses condition shown in Figure 3, top) and three head gestures that we selected from Yan et al. (2018): double tap (nod "Yes" twice), home (head leans toward the right shoulder), and zoom

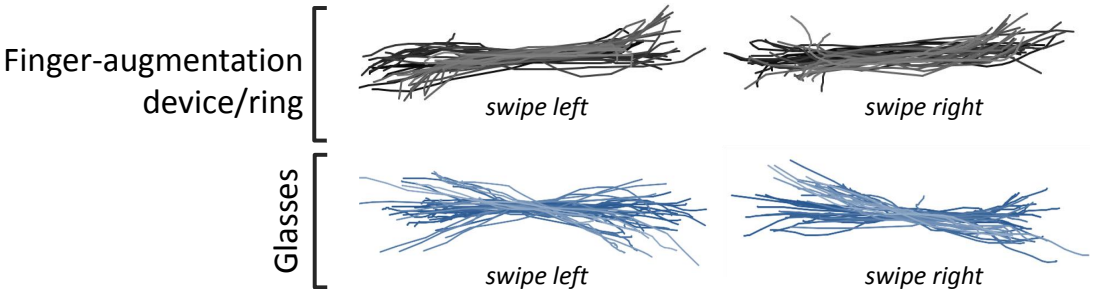

Figure 7. Swipe gestures articulated repeatedly by five participants on a touchscreen worn on the index finger (top) and attached to the temple of a pair of glasses (bottom). *Note:* gesture articulations are shown superimposed (N=5 participants × 8 repetitions = 40 gesture samples per experimental condition).

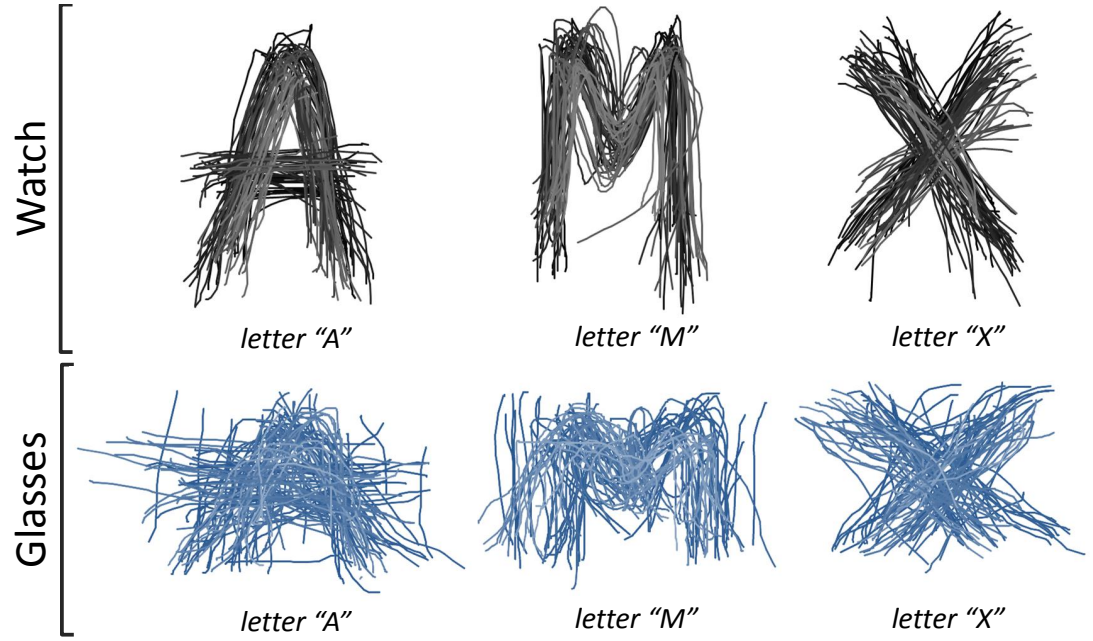

Figure 8. Stroke-gestures articulated repeatedly by six participants on a watch (top) and a touchscreen attached to the temple of a pair of glasses (right). *Note:* gesture articulations are shown superimposed (N=6 participants × 8 repetitions = 48 gesture samples per experimental condition).

in (head leans in front). Figure 9 illustrates the gestures collected repeatedly from five participants, represented as acceleration signals. The figure reveals that zoom-in gestures are both shorter and have less amplitude compared to the "double tap" and "home" gestures. Further investigations at this point could address questions such as: Which head gestures are both fast and accurate? (Hueber et al., 2020), Which head gestures are users willing to perform in public places? (Koelle, Ananthanarayan and Boll, 2020; Rico and Brewster, 2010), and How effective are these gestures for users with upper-body motor impairments? (Cicek, Dave, Feng, Huang, Haines and Nichols, 2020), for example.

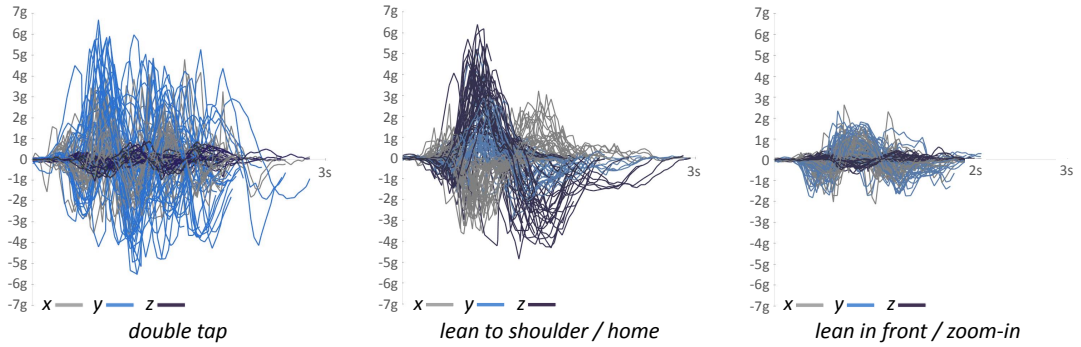

Figure 9. Head gestures performed repeatedly by five participants, illustrated as acceleration signals captured by a device with a built-in three-axis accelerometer attached to the glasses temple. *Note:* gesture articulations are shown superimposed (N=5 participants × 8 repetitions = 40 samples per gesture type).

5. Technical Evaluation of GearWheels

We conducted a controlled simulation experiment to evaluate the technical performance of GearWheels in terms of the efficiency to create events within the software architecture and to transmit corresponding messages with gesture input collected from various types of wearable devices. To this end, we specified another user experiment in GearWheels involving two smartwatches, one smartglasses device, and one smartphone, and we implemented the corresponding InputDevice software components for touch, stroke-gesture, and motion-gesture input, respectively; see the previous section for details regarding these steps.

5.1. Experiment Design

We implemented a repeated-measure, within-subjects design for our technical evaluation experiment with two independent variables:

(1) Message-Size, interval variable representing the size of the messages generated, transmitted, and processed in the GearWheels architecture. We chose six conditions for this variable representing various message sizes arranged in an arithmetic progression with ratio 20 with the following values: 1 KB (*i.e.*, an approximation of the size of the JSON message with metadata only), 21 KB (a one-second long motion gesture or a four-second long stroke-gesture), 41 KB (a two-second motion-gesture or an eight-second stroke-gesture), 61 KB (a three-second motion-gesture), 81 KB (a four-second motion-gesture), and 101 KB (a five-second motion-gesture). These sizes were informed by practical considerations regarding touch, stroke-gesture, and motion-gesture input represented as series of points. For example, the size of our JSON message with metadata only (*e.g.*, information regarding the user, device, test, trial, etc.; see Figure 5) has an upper limit of 1 KB; touch input requires two coordinates, x and y, representing the location of the user's finger on the screen; stroke-gesture input is represented as a series of 2D points with an upper margin of about 5KB for each second of recording;[5] and motion-gestures are represented as series of 9-axis IMU points

[5] An approximation computed with a maximum of 50 bytes per point (*e.g.*, the bytes corresponding to encoding the JSON-formatted string {"X":163, "Y": 422, "ID":0, "T":1325474716230} corresponding to a single touch point (produced by the finger with ID 0) at the screen location (163,422) and the timestamp 1325474716230),

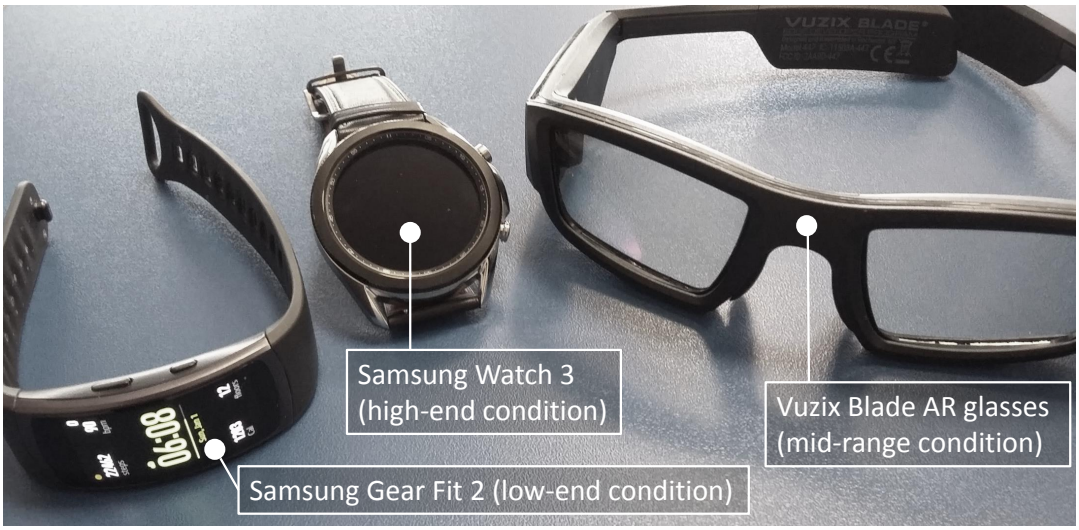

Figure 10. Devices used in our experiment, representative of low-end, mid-range, and high-end wearables.

with an upper message size of 20KB of data per second.[6] For example, the upper limits of the messages employed in the experiments described in Section 4 were 61 KB for stroke-gestures and 101 KB for motion-gestures, respectively.

(2) Input-Device, ordinal variable with four conditions: a low-end wearable (smartwatch), a mid-range wearable (smartglasses), a high-end wearable (smartwatch), and a smartphone (representing our control condition). The characteristics of these devices are presented in detail in Subsection 5.2.

These variables generate a total number of $6 \times 4 = 24$ conditions to measure our dependent variable, Request-Response-Time, representing the time in milliseconds needed by a message to travel from the InputDevice to the Hub via the Link. For each combination of the independent variables, we performed 1,000 repeated measurements denoted as trials in our experiment. To prevent biases caused by data caching in the Wi-Fi network, we randomized the trial order and the content of the messages.

5.2. Apparatus

We ran the Link and Hub software components on a Windows 10 computer (Dell Inspiron 15) featuring an Intel Core i7-7500U 2.7 GHz CPU with 16 GB RAM. We used two smartwatches: Samsung Gear Fit 2 (Tizen 2 operating system, dual-core CPU Exynos 3250, 4 GB internal memory, 512 MB RAM, and a 54 Mbps Wi-Fi connection) representing the low-end condition in our experiment, and Samsung Galaxy Watch 3 (Tizen 5 operating system, dual-core CPU Exynos 9110, 8 GB internal memory, 1 MB RAM, and 54 Mbps Wi-Fi) for the high-end condition. As the mid-range wearable, we used the Vuzix Blade smartglasses (Android 6 operating system, ARM Cortex-A53 CPU, and 2 GB internal memory); see Figure 10. Our control condition was represented by a Huawei P30 Pro smartphone (Android 10 operating system, octa-core CPU Kirin 980, 128 GB internal memory, and 6 GB RAM). The same JavaScript

and 100 points per second result in a total of 5,000 bytes per second.

[6] An approximation computed with a maximum of 200 bytes per point (see footnote 5 for a calculation example for a 2D point) and 100 points per second resulting in a total of 20KB of data per second.

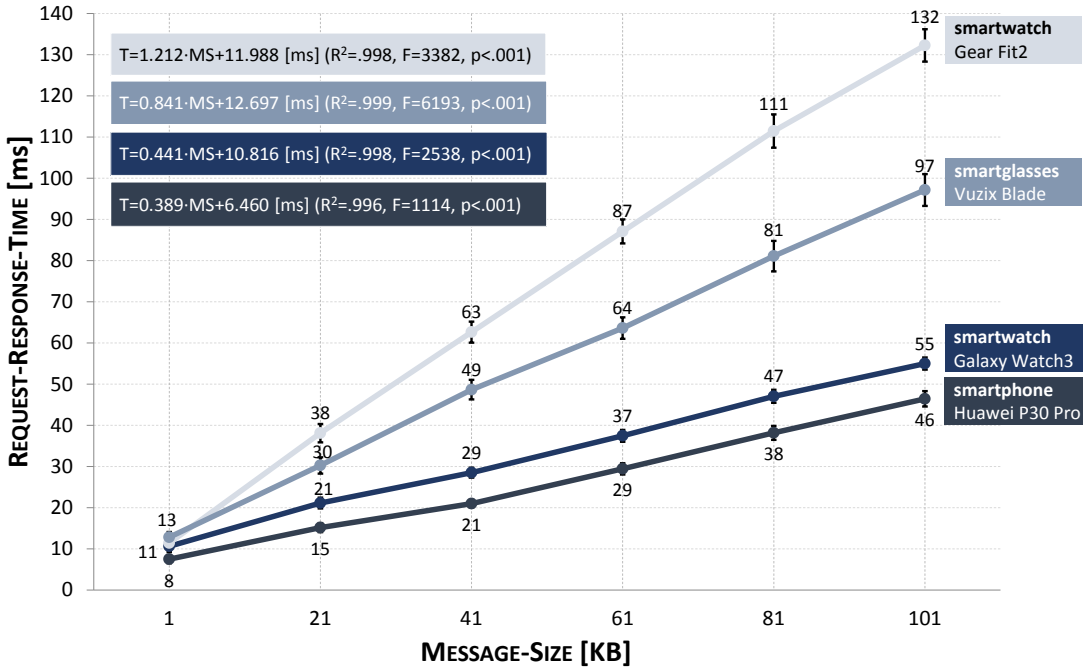

Figure 11. The effect of MESSAGE-SIZE and INPUT-DEVICE on the REQUEST-RESPONSE-TIME to transfer messages of size ranging from 1 KB to 101 KB (a five-second long motion-gesture) using the GearWheels system. *Note:* error bars show 95% CIs.

code (see Figure 6 for an excerpt) ran on each device, despite their different operating systems and hardware specifications. The Wi-Fi network was built around the ASUS RT-AC87U wireless router featuring up to 600 Mbps@2.4GHz.

5.3. Results

Figure 11 illustrates the effect of the MESSAGE-SIZE and DEVICE independent variables on REQUEST-RESPONSE-TIME (average values computed from 1,000 trials). The smallest response time (8 ms) was obtained for messages of 1 KB transferred from the smartphone to the HUB (our control condition). The same device delivered the largest messages of 101 KB with an average response time of 46 ms. Our highest performing wearable, the Samsung Galaxy Watch 3, needed approximately 10 ms more for the same task (55 ms), while the mid-range and low-end devices needed a request-response time that was double (97 ms) and triple (132 ms) in terms of duration, respectively. A regression analysis revealed a statistically significant linear relationship (R^2>.99, p<.001) between REQUEST-RESPONSE-TIME (denoted with "T" in the equations shown in Figure 11) and MESSAGE-SIZE (denoted with "MS") for all the conditions of INPUT-DEVICE. These results (and, especially, a maximum of 132 ms measured for the Gear Fit 2 and the largest message corresponding to a five-second motion gesture) show that the GearWheels software architecture is efficient even for low-end wearables and large message sizes covering gesture input of several seconds. To put these results into the practical perspective of running an experiment with several wearable devices, we refer to the experiments described in Section 4 involving a low-end wearable worn on the finger, at the wrist, and affixed to the temple of a pair of glasses. Our simulation results indicate that it takes up to 132 ms for a motion-gesture

and 87 ms for a stroke-gesture data message to travel in the GearWheels software architecture from the low-end wearable device to the Hub.

6. Limitations and Future Work

GearWheels can be used to support user studies and experiments involving stroke-gestures and motion-gestures performed with wearable devices featuring a web browser or that can natively run JavaScript code. Section 4 showed the versatility of GearWheels for supporting experiments involving various conditions, while Section 5 showed that event and message processing in GearWheels is fast even for low-end wearable devices. Nevertheless, we note several limitations for the current version of GearWheels. Although GearWheels implements support for multi-device scenarios (*i.e.*, multiple devices can be connected at the same time to the Hub component), the current implementation does not allow multiple users to participate simultaneously in the experiment. However, this feature would be useful for studies and experiments where different participants perform gestures in parallel, for example to collaborate during a task. Another limitation refers to the types of gestures that are currently supported by GearWheels, which are represented by touchscreen stroke-gestures and accelerometer/gyroscope motion-gestures only. Other types of gestures, such as whole-body movement, eye gaze gesture input, and others need extensions of GearWheels. Finally, another limitation is that GearWheels does not implement gesture recognition, although such a feature would enable the experimenter to have better control over the gesture data collected during the study, *e.g.*, to recollect gestures that are introduced incorrectly.

There are several opportunities to address these limitations in future work represented by future developments of the GearWheels software. For example, the first limitation requires an updated version of the Hub component so that multiple instances could run at the same time to address gestures produced collaboratively by different participants, *i.e.*, a multi-threaded processing layer added to our software architecture. For GearWheels to accept other types of gestures, other InputDevice components must be implemented, just like we already provide implementations for touchscreen stroke-gesture and accelerometer/gyroscope motion-gesture acquisition. In addition, the Hub must also be updated to handle accordingly the new gesture types. While the former means writing new software modules, the latter is straightforward since the Hub already operates with generic objects. Examples of other gesture types that could be sensed with wearables are on-body, body-referenced, and whole-body gestures for which specific measures of analysis (Vatavu, 2017a) may be interesting for practitioners, eye gaze gestures (Meyer, Schlebusch, Spruit, Hellmig and Kasneci, 2021) detected by smartglasses, or face gestures (Chen, Li, Tao, Lim, Sakashita, Zhang, Guimbretiere and Zhang, 2021) sensed by neck-mounted wearables, to name a few examples. Another useful extension for GearWheels is represented by the incorporation of gesture recognizers, such as open-source approaches (Taranta II, Samiei, Maghoumi, Khaloo, Pittman and LaViola Jr., 2017; Vatavu, 2017b; Vatavu, Anthony and Wobbrock, 2012b; Wobbrock et al., 2007) for feedback to the experimenter during the gesture acquisition phase of the user study or controlled experiment; see also Magrofuoco et al. (2021) for a review of stroke-gesture recognition approaches. Gesture recognizers tailored to the gesture input articulation characteristics of specific user groups, such as users with visual impairments (Vatavu, 2017b) or motor impairments (Vatavu and Ungurean, 2019), may prove useful for

studies and experiments implemented with GearWheels that address topics in gesture input performed with wearables by specific user categories. To enable such developments, we release GearWheels with an open-source license at the web address http://www.eed.usv.ro/mintviz/resources/GearWheels.

7. Conclusion

We introduced in this work a software tool with a dedicated event-based architecture designed to assist researchers and practitioners in their user experiments and studies involving gesture input with wearable devices. Future work will consider an implementation of GearWheels in the form of SaaS/PaaS, but also an extension to accommodate communications with devices via other channels, *e.g.*, Bluetooth, for wearables that do not support Wi-Fi. To the best of our knowledge, GearWheels is the only available tool to support gesture input experiments with wearables, and we look forward to see its utilization in practice towards new developments and scientific discoveries about gesture input with wearable devices.

Acknowledgements

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI-UEFISCDI, project no. PN-III-P2-2.1-PED-2019-0352 (276PED/2020), within PNCDI III.

Notes on contributor(s)

Ovidiu-Andrei Schipor is an Associate Professor at the University of Suceava, Romania. He received the Ph.D. degree in Computer Science from the University of Suceava in 2009, and was awarded a 3-year postdoctoral scholarship (2010–2013) on the topic of intelligent interfaces for verbal communication.

Radu-Daniel Vatavu is a Professor of Computer Science at the University of Suceava, where he directs the Machine Intelligence and Information Visualization Research Laboratory. His research interests include human-computer interaction, augmented reality, ambient intelligence, and accessible computing.

References

Acharya, S., Matovu, R., Serwadda, A., Griswold-Steiner, I., 2019. Gamification of wearable data collection: A tool for both friend and foe, in: Proceedings of the 3rd International Conference on Compute and Data Analysis, pp. 68–77. URL: https://doi.org/10.1145/3314545.3314572, doi:.

Aiordăchioae, A., Schipor, O.A., Vatavu, R.D., 2020. An inventory of voice input commands for users with visual impairments and assistive smartglasses applications, in: Proc. of the International Conference on Development and Application Systems, IEEE. pp. 146–150. URL: https://doi.org/10.1109/DAS49615.2020.9108915, doi:.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I., 2012. Software architecture optimization methods: A systematic literature review. IEEE Transactions on Software Engineering 39, 658–683. URL: https://doi.org/10.1109/TSE.2012.64, doi:.

Ali, A.X., Morris, M.R., Wobbrock, J.O., 2019. Crowdlicit: A system for conducting distributed end-user elicitation and identification studies, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp. 1–12. URL: https://doi.org/10.1145/3290605.3300485, doi:.

Alpers, S., Benţa, K.I., 2021. Sensegraph: Affect self-monitoring and tagging tool with wearable devices, in: 2021 International Conference on Computer Communications and Networks (ICCCN), IEEE. pp. 1–6.

Anthony, L., Vatavu, R.D., Wobbrock, J.O., 2013. Understanding the consistency of users' pen and finger stroke gesture articulation, in: Proceedings of Graphics Interface 2013, Canadian Information Processing Society, CAN. p. 87–94. URL: https://dl.acm.org/doi/10.5555/2532129.2532145.

Appert, C., Zhai, S., 2009. Using strokes as command shortcuts: Cognitive benefits and toolkit support, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 2289–2298. URL: https://doi.org/10.1145/1518701.1519052, doi:.

Arcos-Medina, G., Menéndez, J., Vallejo, J., 2018. Comparative study of performance and productivity of mvc and mvvm design patterns. KnE Engineering , 241–252URL: https://doi.org/10.18502/keg.v1i2.1498, doi:.

Ashbrook, D., Starner, T., 2010. Magic: a motion gesture design tool, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2159–2168. URL: https://doi.org/10.1145/1753326.1753653, doi:.

Ast, M., Gaedke, M., 2017. Self-contained web components through serverless computing, in: Proc. of the 2nd International Workshop on Serverless Computing, pp. 28–33. URL: https://doi.org/10.1145/3154847.3154849, doi:.

Baber, C., 2001. Wearable computers: A human factors review. International Journal of Human–Computer Interaction 13, 123–145. URL: https://doi.org/10.1207/S15327590IJHC1302_3, doi:.

Bailly, G., Müller, J., Rohs, M., Wigdor, D., Kratz, S., 2012. Shoesense: A new perspective on gestural interaction and wearable applications, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. pp. 1239–1248. URL: http://doi.acm.org/10.1145/2207676.2208576, doi:.

Banijamali, A., Pakanen, O.P., Kuvaja, P., Oivo, M., 2020. Software architectures of the convergence of cloud computing and the internet of things: A systematic literature review. Information and Software Technology 122, 106271. URL: https://doi.org/10.1016/j.infsof.2020.106271, doi:.

Bass, L., Clements, P., Kazman, R., 2003. Software architecture in practice. Addison-Wesley Professional. URL: https://dl.acm.org/doi/book/10.5555/773239, doi:.

Beaudouin-Lafon, M., Mackay, W., 2002. Prototyping Tools and Techniques, in: The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications. L. Erlbaum Assoc. Inc., USA, pp. 1006–1031.

Boldu, R., Dancu, A., Matthies, D.J., Cascón, P.G., Ransir, S., Nanayakkara, S., 2018. Thumb-in-motion: Evaluating thumb-to-ring microgestures for athletic activity, in: Proceedings of the Symposium on Spatial User Interaction, Association for Computing Machinery, New York, NY, USA. p. 150–157. URL: https://doi.org/10.1145/3267782.3267796.

Bomsdorf, B., Blum, R., Künkel, D., 2017. Towards ProGesture, a Tool Supporting Early Prototyping of 3D-Gesture Interaction, in: 3D Printing: Breakthroughs in Research and Practice. IGI Global, pp. 396–413. URL: https://doi.org/10.4018/IJPOP.2015070103.

Breivold, H.P., Crnkovic, I., Larsson, M., 2012. A systematic review of software architecture evolution research. Information and Software Technology 54, 16–40. URL: https://doi.org/10.1016/j.infsof.2011.06.002, doi:.

Brudy, F., Holz, C., Rädle, R., Wu, C.J., Houben, S., Klokmose, C.N., Marquardt, N., 2019. Cross-device taxonomy: Survey, opportunities and challenges of interactions spanning across multiple devices, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–28. URL: https://doi.org/10.1145/

3290605.3300792, doi:.
Buruk, O.T., Özcan, O., 2017. Gestanalytics: Experiment and analysis tool for gesture-elicitation studies, in: Proceedings of the 2017 ACM Conference Companion Publication on Designing Interactive Systems, pp. 34–38. URL: https://doi.org/10.1145/3064857.3079114, doi:.
Buzzi, M.C., Buzzi, M., Leporini, B., Trujillo, A., 2017. Analyzing visually impaired people's touch gestures on smartphones. Multimedia Tools Appl. 76, 5141–5169. URL: https://doi.org/10.1007/s11042-016-3594-9.
Cao, X., Zhai, S., 2007. Modeling human performance of pen stroke gestures, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1495–1504. URL: https://doi.org/10.1145/1240624.1240850.
Chandy, K.M., 2006. Event-driven applications: Costs, benefits and design approaches. Gartner Application Integration and Web Services Summit 2006. URL: https://doi.org/10.1007/978-0-387-39940-9_570, doi:.
Chen, T., Li, Y., Tao, S., Lim, H., Sakashita, M., Zhang, R., Guimbretiere, F., Zhang, C., 2021. Neckface: Continuously tracking full facial expressions on neck-mounted wearables. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 5. URL: https://doi.org/10.1145/3463511, doi:.
Chen, W.H., 2015. Blowatch: Blowable and hands-free interaction for smartwatches, in: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, USA. pp. 103–108. URL: http://doi.acm.org/10.1145/2702613.2726961, doi:.
Chen, Z., Ma, X., Peng, Z., Zhou, Y., Yao, M., Ma, Z., Wang, C., Gao, Z., Shen, M., 2018. User-defined gestures for gestural interaction: Extending from hands to other body parts. International Journal of Human–Computer Interaction 34, 238–250. URL: https://doi.org/10.1080/10447318.2017.1342943, doi:.
Cicek, M., Dave, A., Feng, W., Huang, M.X., Haines, J.K., Nichols, J., 2020. Designing and evaluating head-based pointing on smartphones for people with motor impairments, in: Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility, Association for Computing Machinery, New York, NY, USA. pp. 14:1–14:12. URL: https://doi.org/10.1145/3373625.3416994.
Cioată, P.V., Vatavu, R.D., 2018. 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, ACM, New York, NY, USA. pp. 60:1–60:2. URL: http://doi.acm.org/10.1145/3180308.3180369, doi:.
Clements, P., Garlan, D., Little, R., Nord, R., Stafford, J., 2003. Documenting software architectures: views and beyond, in: 25th International Conference on Software Engineering, 2003. Proceedings., IEEE. pp. 740–741. URL: https://dl.acm.org/doi/10.5555/776816.776928, doi:.
Cruz-Benito, J., Garcia-Penalvo, F.J., Theron, R., 2019. Analyzing the software architectures supporting HCI/HMI processes through a systematic review of the literature. Telematics and Informatics 38, 118–132. URL: https://doi.org/10.1016/j.tele.2018.09.006, doi:.
Cui, X., Sun, Y., Xiao, S., Mei, H., 2009. Architecture design for the large-scale software-intensive systems: A decision-oriented approach and the experience, in: Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems, IEEE. pp. 30–39. URL: https://doi.org/10.1109/ICECCS.2009.42, doi:.
Dingler, T., Rzayev, R., Shirazi, A.S., Henze, N., 2018. Designing consistent gestures across device types: Eliciting rsvp controls for phone, watch, and glasses, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–12. URL: https://doi.org/10.1145/3173574.3173993.
Endow, S., Moradi, H., Srivastava, A., Noya, E.G., Torres, C., 2021. Compressables: A haptic prototyping toolkit for wearable compression-based interfaces, in: Designing Interactive Systems Conference 2021, pp. 1101–1114.
Esteves, A., Velloso, E., Bulling, A., Gellersen, H., 2015. Orbits: Gaze interaction for smart

watches using smooth pursuit eye movements, in: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, ACM, New York, NY, USA. pp. 457–466. URL: http://doi.acm.org/10.1145/2807442.2807499, doi:.

Evesti, A., 2007. Quality-oriented software architecture development. VTT Technical Research Centre of Finland URL: http://www.vtt.fi/inf/pdf/publications/2007/P636.pdf.

Folmer, E., van Gurp, J., Bosch, J., 2003. Scenario-based assessment of software architecture usability., in: ICSE Workshop on SE-HCI, Citeseer. pp. 61–68.

García-Holgado, A., García-Peñalvo, F.J., 2016. Architectural pattern to improve the definition and implementation of elearning ecosystems. Science of Computer Programming 129, 20–34. URL: https://doi.org/10.1016/j.scico.2016.03.010, doi:.

Gheran, B.F., Vanderdonckt, J., Vatavu, R.D., 2018. Gestures for smart rings: Empirical results, insights, and design implications, in: Proceedings of the 2018 Designing Interactive Systems Conference, ACM, New York, NY, USA. pp. 623–635. URL: http://doi.acm.org/10.1145/3196709.3196741, doi:.

Gheran, B.F., Villarreal-Narvaez, S., Vatavu, R.D., Vanderdonckt, J., 2022. Repliges and gestory: Visual tools for systematizing and consolidating knowledge on user-defined gestures, in: Proceedings of 30th International Conference on Advanced Visual Interfaces, ACM, New York, NY, USA.

Gherman, O., Schipor, O., Gheran, B.F., 2018. Verge: A system for collecting voice, eye gaze, gesture, and eeg data for experimental studies, in: 2018 International Conference on Development and Application Systems (DAS), IEEE. pp. 150–155. URL: https://doi.org/10.1109/DAAS.2018.8396088, doi:.

Grossman, T., Chen, X.A., Fitzmaurice, G., 2015. Typing on glasses: Adapting text entry to smart eyewear, in: Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, ACM, New York, NY, USA. p. 144–152. URL: https://doi.org/10.1145/2785830.2785867.

Haladjian, J., 2019. The wearables development toolkit: an integrated development environment for activity recognition applications. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 3, 1–26.

Hauck, S., 1995. Asynchronous design methodologies: An overview. Proceedings of the IEEE 83, 69–93. URL: https://doi.org/10.1109/5.362752, doi:.

Heller, F., Ivanov, S., Wacharamanotham, C., Borchers, J., 2014. Fabritouch: Exploring flexible touch input on textiles, in: Proceedings of the 2014 ACM International Symposium on Wearable Computers, ACM, New York, NY, USA. p. 59–62. URL: https://doi.org/10.1145/2634317.2634345.

Hewett, T.T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., Verplank, W., 1992. ACM SIGCHI curricula for human-computer interaction. ACM. URL: https://doi.org/10.1145/2594128, doi:.

Hueber, S., Cherek, C., Wacker, P., Borchers, J., Voelker, S., 2020. Headbang: Using head gestures to trigger discrete actions on mobile devices, in: Proceedings of the 22nd International Conference on Human-Computer Interaction with Mobile Devices and Services, ACM, New York, NY, USA. URL: https://doi.org/10.1145/3379503.3403538, doi:.

International Organization for Standarization, 2005. IEC 25000 Software and system engineering–Software product Quality Requirements and Evaluation (SQuaRE)–Guide to SQuaRE. URL: https://www.iso.org/standard/35683.html.

Jang, S., Elmqvist, N., Ramani, K., 2014. Gestureanalyzer: visual analytics for pattern analysis of mid-air hand gestures, in: Proceedings of the 2nd ACM symposium on Spatial user interaction, pp. 30–39. URL: https://doi.org/10.1145/2659766.2659772, doi:.

Jones, L., Nabil, S., McLeod, A., Girouard, A., 2020. Wearable bits: scaffolding creativity with a prototyping toolkit for wearable e-textiles, in: Proc. of the 14th International Conference on Tangible, Embedded, and Embodied Interaction, pp. 165–177. URL: https://doi.org/10.1145/3374920.3374954, doi:.

Jones, R., Milic-frayling, N., Rodden, K., Blackwell, A., 2007. Contextual method for the redesign of existing software products. International Journal of Human–Computer

Interaction 22, 81–101. URL: https://www.tandfonline.com/doi/abs/10.1080/10447310709336956, doi:.

Kazemitabaar, M., McPeak, J., Jiao, A., He, L., Outing, T., Froehlich, J.E., 2017. Makerwear: A tangible approach to interactive wearable creation for children, in: Proc. of the 2017 Conference on Human Factors in Computing Systems, pp. 133–145. URL: https://doi.org/10.1145/3025453.3025887, doi:.

Kazman, R., Bass, L., Abowd, G., Webb, M., 1994. Saam: A method for analyzing the properties of software architectures, in: Proceedings of 16th International Conference on Software Engineering, IEEE. pp. 81–90. URL: https://dl.acm.org/doi/abs/10.5555/257734.257746, doi:.

Kim, D., Kwon, J., Han, S., Park, Y.L., Jo, S., 2018. Deep full-body motion network for a soft wearable motion sensing suit. IEEE/ASME Transactions on Mechatronics 24, 56–66. URL: https://doi.org/10.1109/TMECH.2018.2874647, doi:.

Kin, K., Hartmann, B., DeRose, T., Agrawala, M., 2012a. Proton++ a customizable declarative multitouch framework, in: Proceedings of the 25th annual ACM symposium on User interface software and technology, pp. 477–486. URL: https://doi.org/10.1145/2380116.2380176, doi:.

Kin, K., Hartmann, B., DeRose, T., Agrawala, M., 2012b. Proton: multitouch gestures as regular expressions, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2885–2894. URL: https://doi.org/10.1145/2207676.2208694, doi:.

Koelle, M., Ananthanarayan, S., Boll, S., 2020. Social acceptability in hci: A survey of methods, measures, and design strategies, in: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–19. URL: https://doi.org/10.1145/3313831.3376162, doi:.

Kohlsdorf, D., Starner, T., Ashbrook, D., 2011. Magic 2.0: A web tool for false positive prediction and prevention for gesture recognition systems, in: Face and Gesture 2011, IEEE. pp. 1–6. URL: https://doi.org/10.1109/FG.2011.5771412, doi:.

Kruchten, P., Obbink, H., Stafford, J., 2006. The past, present, and future for software architecture. IEEE software 23, 22–30. URL: https://doi.org/10.1109/MS.2006.59, doi:.

Kubo, Y., Shizuki, B., Tanaka, J., 2016. B2b-swipe: Swipe gesture for rectangular smartwatches from a bezel to a bezel, in: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 3852–3856. URL: https://doi.org/10.1145/2858036.2858216.

Leiva, L.A., Martín-Albo, D., Plamondon, R., 2015. Gestures à Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. ACM Trans. Intell. Syst. Technol. 7. URL: https://doi.org/10.1145/2799648, doi:.

Leiva, L.A., Martín-Albo, D., Plamondon, R., Vatavu, R.D., 2018. Keytime: Super-accurate prediction of stroke gesture production times, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–12. URL: https://doi.org/10.1145/3173574.3173813.

Leiva, L.A., Vatavu, R.D., Martin-Albo, D., Plamondon, R., 2020. Omnis prædictio: Estimating the full spectrum of human performance with stroke gestures. International Journal of Human-Computer Studies 142, 102466. URL: https://doi.org/10.1016/j.ijhcs.2020.102466.

Long, A.C., Landay, J.A., Rowe, L.A., Michiels, J., 2000. Visual similarity of pen gestures, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 360–367. URL: https://doi.org/10.1145/332040.332458, doi:.

Long Jr, A.C., Landay, J.A., Rowe, L.A., 1999. Implications for a gesture design tool, in: Proc. of the SIGCHI Conf. on Human Factors in Computing Systems, pp. 40–47. URL: https://doi.org/10.1145/302979.302985, doi:.

Lopes, P., Ion, A., Mueller, W., Hoffmann, D., Jonell, P., Baudisch, P., 2015. Proprioceptive Interaction, in: CHI '15, pp. 939–948. URL: https://doi.org/10.1145/2702123.2702461.

Lü, H., Fogarty, J.A., Li, Y., 2014. Gesture script: Recognizing gestures and their structure using rendering scripts and interactively trained parts, in: Proceedings of the SIGCHI

Conference on Human Factors in Computing Systems, pp. 1685–1694. URL: https://doi.org/10.1145/2556288.2557263, doi:.

Lü, H., Li, Y., 2012. Gesture coder: a tool for programming multi-touch gestures by demonstration, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2875–2884. URL: https://doi.org/10.1145/2207676.2208693, doi:.

Magrofuoco, N., Roselli, P., Vanderdonckt, J., 2021. Two-dimensional stroke gesture recognition: A survey. ACM Comput. Surv. 54. URL: https://doi.org/10.1145/3465400, doi:.

Magrofuoco, N., Roselli, P., Vanderdonckt, J., Pérez-Medina, J.L., Vatavu, R.D., 2019. Gestman: a cloud-based tool for stroke-gesture datasets, in: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 1–6. URL: https://doi.org/10.1145/3319499.3328227, doi:.

Magrofuoco, N., Vanderdonckt, J., 2019. Gelicit: A cloud platform for distributed gesture elicitation studies. Proceedings of the ACM on Human-Computer Interaction 3, 1–41. URL: https://doi.org/10.1145/3331148, doi:.

Malu, M., Chundury, P., Findlater, L., 2018. Exploring accessible smartwatch interactions for people with upper body motor impairments, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–12. URL: https://doi.org/10.1145/3173574.3174062.

Malu, M., Findlater, L., 2015. Personalized, wearable control of a head-mounted display for users with upper body motor impairments, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 221–230. URL: https://doi.org/10.1145/2702123.2702188.

Meyer, J., Schlebusch, T., Spruit, H., Hellmig, J., Kasneci, E., 2021. A novel gaze gesture sensor for smart glasses based on laser self-mixing, in: Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. URL: https://doi.org/10.1145/3411763.3451621.

Moxey, C., Edwards, M., Etzion, O., Ibrahim, M., Iyer, S., Lalanne, H., Monze, M., Peters, M., Rabinovich, Y., Sharon, G., et al., 2010. A conceptual model for event processing systems. IBM Redguide publication doi:.

Myers, B.A., 1995. User Interface Software Tools. ACM Trans. Comput.-Hum. Interact. 2, 64–103. URL: https://doi.org/10.1145/200968.200971.

Narayana, P., Beveridge, R., Draper, B.A., 2018. Gesture recognition: Focus on the hands, in: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5235–5244. URL: https://doi.org/10.1109/CVPR.2018.00549, doi:.

Nebeling, M., Ott, D., Norrie, M.C., 2015. Kinect analysis: a system for recording, analysing and sharing multimodal interaction elicitation studies, in: Proc. of the 7th ACM Symposium on Engineering Interactive Computing Systems, pp. 142–151. URL: https://doi.org/10.1145/2774225.2774846, doi:.

Nebeling, M., Teunissen, E., Husmann, M., Norrie, M.C., 2014. Xdkinect: Development framework for cross-device interaction using kinect, in: Proc. of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, ACM, New York, NY, USA. pp. 65–74. URL: http://doi.acm.org/10.1145/2607023.2607024, doi:.

Oh, U., Findlater, L., 2014. Design of and subjective response to on-body input for people with visual impairments, in: Proc. of the 16th Int. ACM SIGACCESS Conference on Computers & Accessibility, ACM, New York, NY, USA. p. 115–122. URL: https://doi.org/10.1145/2661334.2661376.

Poppinga, B., Sahami Shirazi, A., Henze, N., Heuten, W., Boll, S., 2014. Understanding shortcut gestures on mobile touch devices, in: Proceedings of the 16th International Conference on Human-Computer Interaction with Mobile Devices & Services, ACM, New York, NY, USA. p. 173–182. URL: https://doi.org/10.1145/2628363.2628378.

Reinecke, K., Gajos, K.Z., 2015. LabintheWild: Conducting large-scale online experiments with uncompensated samples, in: Proceedings of the 18th ACM conference on computer supported cooperative work & social computing, pp. 1364–1378. URL: https://doi.org/

10.1145/2675133.2675246, doi:.

Rico, J., Brewster, S., 2010. Usable gestures for mobile interfaces: Evaluating social acceptability, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 887–896. URL: https://doi.org/10.1145/1753326.1753458, doi:.

Roda, C., Navarro, E., Zdun, U., López-Jaquero, V., Simhandl, G., 2018. Past and future of software architectures for context-aware systems: A systematic mapping study. Journal of Systems and Software 146, 310–355. URL: https://doi.org/10.1016/j.jss.2018.09.074, doi:.

Roggen, D., 2020. ARM cortex M4-based extensible multimodal wearable platform for sensor research and context sensing from motion & sound, in: Adjunct Proc. of the 2020 ACM Int. Joint Conference on Pervasive and Ubiquitous Computing and Proc. of the 2020 ACM Int. Symposium on Wearable Computers, pp. 284–289. URL: https://doi.org/10.1145/3410530.3414368, doi:.

Roy, Q., Malacria, S., Guiard, Y., Lecolinet, E., Eagan, J., 2013. Augmented letters: Mnemonic gesture-based shortcuts, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 2325–2328. URL: https://doi.org/10.1145/2470654.2481321.

Schipor, O.A., Bilius, L.B., Vatavu, R.D., 2022. Wearskill: Personalized and interchangeable input with wearables for users with motor impairments, in: Proceedings of the 19th Web for All Conference, ACM, New York, NY, USA. URL: https://doi.org/10.1145/3493612.3520455.

Schipor, O.A., Pentiuc, S.G., Schipor, M.D., 2011. Towards a multimodal emotion recognition framework to be integrated in a computer based speech therapy system, in: Proc. of the 6th Conference on Speech Technology and Human-Computer Dialogue, IEEE. pp. 1–6. URL: https://doi.org/10.1109/SPED.2011.5940727, doi:.

Schipor, O.A., Pentiuc, S.G., Schipor, M.D., 2012. Toward automatic recognition of children's affective state using physiological parameters and fuzzy model of emotions. Advances in Electrical and Computer Engineering 12, 47–50. URL: https://doi.org/10.4316/aece.2012.02008, doi:.

Schipor, O.A., Vatavu, R.D., Vanderdonckt, J., 2019a. Euphoria: A scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments. Information and Software Technology 109, 43–59. URL: https://doi.org/10.1016/j.infsof.2019.01.006, doi:.

Schipor, O.A., Vatavu, R.D., Wu, W., 2019b. Sapiens: Towards software architecture to support peripheral interaction in smart environments. Proceedings of the ACM on Human-Computer Interaction 3, 1–24. URL: https://doi.org/10.1145/3331153, doi:.

Schipor, O.A., Wu, W., Tsai, W.T., Vatavu, R.D., 2017. Software architecture design for spatially-indexed media in smart environments. Advances in Electrical and Computer Engineering 17, 17–23. URL: https://doi.org/10.4316/AECE.2017.02003, doi:.

Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F., 2013. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. volume 2. John Wiley & Sons, West Sussex, England.

Siean, A.I., Vatavu, R.D., 2021. Wearable interactions for users with motor impairments: Systematic review, inventory, and research implications, in: Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility, ACM, New York, NY, USA. URL: https://doi.org/10.1145/3441852.3471212, doi:.

Singh, M., Fernandez-Luque, L., Srivastava, J., 2017. The 360qs toolkit for sleep and physical activity analysis based on wearables, in: 2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS), IEEE. pp. 680–681.

Solis, R., Pakbin, A., Akbari, A., Mortazavi, B.J., Jafari, R., 2019. A human-centered wearable sensing platform with intelligent automated data annotation capabilities, in: Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 255–260. URL: https://doi.org/10.1145/3302505.3310087, doi:.

Tănase, C.A., Vatavu, R.D., Pentiuc, Ş.G., Graur, A., 2008. Detecting and tracking multiple users in the proximity of interactive tabletops. Advances in Electrical and Computer Engineering 8, 61–64. URL: https://doi.org/10.4316/AECE.2008.02011, doi:.

Tang, J.K., Igarashi, T., 2013. Cubod: a customized body gesture design tool for end users, in: Proc. of the 27th International BCS Human Computer Interaction Conference, pp. 1–10. URL: https://doi.org/10.14236/ewic/HCI2013.22, doi:.

Taranta II, E.M., Samiei, A., Maghoumi, M., Khaloo, P., Pittman, C.R., LaViola Jr., J.J., 2017. Jackknife: A reliable recognizer with few samples and many modalities, in: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. p. 5850–5861. URL: https://doi.org/10.1145/3025453.3026002, doi:.

Terenti, M., Vatavu, R.D., 2021. How do hci researchers describe their software tools? insights from a synopsis survey of tools for multimodal interaction, in: Companion Publication of the 2021 International Conference on Multimodal Interaction, ACM, New York, NY, USA. p. 7–12. URL: https://doi.org/10.1145/3461615.3485431, doi:.

Ungurean, C., Pentiuc, S.G., Vatavu, R.D., 2009. Use your head: an interface for computer games using head gestures, in: 8th International Gesture Workshop (GW'09), Gesture in Embodied Communication and Human-Computer Interaction 2009. URL: https://doi.org/10.1109/RATFG.2001.938911, doi:.

Vanderdonckt, J., Zen, M., Vatavu, R.D., 2019. AB4Web: An On-Line A/B Tester for Comparing User Interface Design Alternatives. Proc. of the ACM on Human-Computer Interaction 3. URL: https://doi.org/10.1145/3331160.

Vatavu, R.D., 2012. Nomadic gestures: A technique for reusing gesture commands for frequent ambient interactions. J. Ambient Intell. Smart Environ. 4, 79–93. URL: http://dl.acm.org/citation.cfm?id=2350758.2350765.

Vatavu, R.D., 2017a. Beyond features for recognition: Human-readable measures to understand users' whole-body gesture performance. International Journal of Human–Computer Interaction 33, 713–730. URL: https://doi.org/10.1080/10447318.2017.1278897, doi:.

Vatavu, R.D., 2017b. Improving gesture recognition accuracy on touch screens for users with low vision, in: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. p. 4667–4679. URL: https://doi.org/10.1145/3025453.3025941, doi:.

Vatavu, R.D., 2017c. Smart-pockets: Body-deictic gestures for fast access to personal data during ambient interactions. Int. J. of Human-Computer Studies 103, 1–21. URL: https://doi.org/10.1016/j.ijhcs.2017.01.005, doi:.

Vatavu, R.D., 2019. The dissimilarity-consensus approach to agreement analysis in gesture elicitation studies, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–13. URL: https://doi.org/10.1145/3290605.3300454.

Vatavu, R.D., Anthony, L., Wobbrock, J.O., 2012a. Gestures as point clouds: a $ p recognizer for user interface prototypes, in: Proceedings of the 14th ACM international conference on Multimodal interaction, pp. 273–280. URL: https://doi.org/10.1145/2388676.2388732, doi:.

Vatavu, R.D., Anthony, L., Wobbrock, J.O., 2012b. Gestures as point clouds: A $p recognizer for user interface prototypes, in: Proceedings of the 14th ACM International Conference on Multimodal Interaction, ACM, New York, NY, USA. p. 273–280. URL: https://doi.org/10.1145/2388676.2388732, doi:.

Vatavu, R.D., Anthony, L., Wobbrock, J.O., 2013. Relative accuracy measures for stroke gestures, in: Proceedings of the 15th ACM on International conference on multimodal interaction, pp. 279–286. URL: https://doi.org/10.1145/2522848.2522875, doi:.

Vatavu, R.D., Cramariuc, G., Schipor, D.M., 2015. Touch interaction for children aged 3 to 6 years: Experimental findings and relationship to motor skills. International Journal of Human-Computer Studies 74, 54–76. URL: http://dx.doi.org/10.1016/j.ijhcs.2014.10.007.

Vatavu, R.D., Gheran, B.F., Schipor, M.D., 2018. The impact of low vision on touch-gesture articulation on mobile devices. IEEE Pervasive Computing 17, 27–37. URL: https://doi.org/10.1109/MPRV.2018.011591059, doi:.

Vatavu, R.D., Ungurean, O.C., 2019. Stroke-gesture input for people with motor impairments: Empirical results & research roadmap, in: Proc. of the CHI Conf. on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1–14. URL: https://doi.org/10.1145/3290605.3300445.

Vatavu, R.D., Ungurean, O.C., 2022. Gesture input articulation with upper-body wearables for users with upper-body motor impairments, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. doi:.

Vatavu, R.D., Vogel, D., Casiez, G., Grisoni, L., 2011. Estimating the perceived difficulty of pen gestures, in: Proceedings of the 13th IFIP TC 13 International Conference on Human-Computer Interaction - Volume Part II, Springer-Verlag, Berlin, Heidelberg. p. 89–106. URL: https://doi.org/10.1007/978-3-642-23771-3_9.

Vatavu, R.D., Wobbrock, J.O., 2015. Formalizing agreement analysis for elicitation studies: New measures, significance test, and toolkit, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1325–1334. URL: https://doi.org/10.1145/2702123.2702223.

Villarreal-Narvaez, S., Vanderdonckt, J., Vatavu, R.D., Wobbrock, J.O., 2020. A systematic review of gesture elicitation studies: What can we learn from 216 studies?, in: Proceedings of the 2020 ACM Designing Interactive Systems Conference, ACM, New York, NY, USA. p. 855–872. URL: https://doi.org/10.1145/3357236.3395511.

Vuletic, T., Duffy, A., Hay, L., McTeague, C., Campbell, G., Grealy, M., 2019. Systematic literature review of hand gestures used in human computer interaction interfaces. International Journal of Human-Computer Studies 129, 74–94. URL: https://doi.org/10.1016/j.ijhcs.2019.03.011, doi:.

Wobbrock, J.O., Kientz, J.A., 2016. Research Contributions in Human-Computer Interaction. Interactions 23, 38–44. URL: https://doi.org/10.1145/2907069.

Wobbrock, J.O., Morris, M.R., Wilson, A.D., 2009. User-defined gestures for surface computing, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. p. 1083–1092. URL: https://doi.org/10.1145/1518701.1518866.

Wobbrock, J.O., Wilson, A.D., Li, Y., 2007. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes, in: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, Association for Computing Machinery, New York, NY, USA. p. 159–168. URL: https://doi.org/10.1145/1294211.1294238.

Wu, H., Yang, L., 2020. User-defined gestures for dual-screen mobile interaction. International Journal of Human–Computer Interaction 36, 978–992. URL: https://doi.org/10.1080/10447318.2019.1706331, doi:.

Yamada, S., Kakue, T., Shimobaba, T., Ito, T., 2018. Interactive holographic display based on finger gestures. Scientific Reports 8, 1–7. URL: https://doi.org/10.1038/s41598-018-20454-6, doi:.

Yan, Y., Shi, Y., Yu, C., Shi, Y., 2020. Headcross: Exploring head-based crossing selection on head-mounted displays. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4. URL: https://doi.org/10.1145/3380983.

Yan, Y., Yu, C., Yi, X., Shi, Y., 2018. Headgesture: Hands-free input approach leveraging head movements for hmd devices. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2. URL: https://doi.org/10.1145/3287076.

Zaiţi, I.A., Pentiuc, Ş.G., Vatavu, R.D., 2015. On free-hand TV control: experimental results on user-elicited gestures with Leap Motion. Personal and Ubiquitous Computing 19, 821–838. URL: https://doi.org/10.1007/s00779-015-0863-y, doi:.