A Mulsemedia Framework for Delivering Sensory Effects to Heterogeneous Systems[1]

Estêvão Bissoli Saleme
Celso A. S. Santos
Gheorghita Ghinea

Received: date / Accepted: date

Technological advances in computing have allowed multimedia systems to create more immersive experiences for users. Beyond the traditional senses of sight and hearing, researchers have observed that the use of smell, taste, and touch in such systems is becoming increasingly well-received, leading to a new category of multimedia systems called mulsemedia—multiple sensorial media—systems. In parallel, these systems introduce heterogeneous technologies to deliver different sensory effects such as lighting, wind, vibration, and smell, under varied conditions and restrictions. This new paradigm shift poses many challenges, mainly related to mulsemedia integration, delay, responsiveness, sensory effects intensities, wearable and other heterogeneous devices for delivering sensory effects, and remote delivery of mulsemedia components. In addition, new approaches to interacting with multimedia applications have emerged such as multi-touch interfaces, voice processing, and brain-computer interfaces, giving rise to new kinds of complex interactive systems. In this article, we underpin fundamental challenges for delivering multisensory effects to heterogeneous systems. We propose an interoperable mulsemedia framework for coping with these challenges, meeting the emerging requirements. It is achieved through the evolution of an open distributed mulsemedia system. We changed its core following architectural and design patterns to accommodate different profiles of communication, connectivity, and sensory effects metadata standard according to the need of mulsemedia applications and devices available in the user's environment. The results include case studies where the framework has been duly applied.

Keywords
Mulsemedia systems; Multimedia Applications; Interoperability; Software Integration; Frameworks.

1 Introduction

Mulsemedia [19] systems feature heterogeneous technologies to deliver a variety of sensory effects such as lighting, wind, vibration, and smell, under varied conditions and restrictions [47]. Aside from digital multisensory systems, which do not necessarily involve multimedia, mulsemedia aggregates the challenges of dealing with audiovisual content along with at least one more sensory effect mainly drawn from those of smell, touch, and taste. Whilst aiming at reuse in this domain, the variability of its complex ecosystem, which involves interweaving different software and hardware to render these sensory effects, imposes laborious and challenging tasks. System variability is related to being customized for specific needs. Although there is a lack of consensus with regard to this particular meaning, this term has been used among practitioners and researchers alike to denote variation and changes in

[1] This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Finance Code 88881.187844/2018-01. Prof. C. A. S. Santos was supported by a research grant from the Brazilian National Council for Scientific and Technological Development (CNPq) (#312148/2014-3) and (Fundação de Amparo à Pesquisa e Inovação do Espírito Santo) FAPES (#67927378/2015). Prof. G. Ghinea gratefully acknowledges funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 688503 for the NEWTON project (http://www.newtonproject.eu). E. B. Saleme additionally acknowledges support from the Federal Institute of Espírito Santo.

A Mulsemedia Framework for Delivering Sensory Effects to Heterogeneous Systems
E. B. Saleme, C. A. S. Santos, Federal University of Espírito Santo, Av. Fernando Ferrari, 514, 29075-910 - Vitória-ES, Brazil
Tel.: +55 27 4009-2324, Fax: +55 27 4009-2850
estevaobissoli@gmail.com, saibel@inf.ufes.br
G. Ghinea, Brunel University London, Wilfred Brown Building 215, Kingston Lane, Uxbridge, Middlesex UB8 3PH
Tel.: +44 (0)1895 274000, Fax: +44 (0)1895 232806
george.ghinea@brunel.ac.uk

architecture to support different scenarios of usage [21, 17, 36].

An example to illustrate variability in mulsemedia systems is when users wish to use different rendering devices and multimedia applications for the same purpose because they are not satisfied with some feature in them or just because others present new characteristics craved by them. This change is not straightforward in the realm of mulsemedia systems mostly because the solutions presented in the literature were devised for specific purposes other than that of multiple reuse. Therefore, devices designed for the same purpose are seldom interchangeable. One way to deal with variability is through interoperability—the cooperation between two or more software components that could be designed in distinct languages, interfaces, and execution platforms [63]. The European Committee for Interoperable Systems (ECIS) states that a device that cannot work in conjunction with other IT (Information Technology) products which consumers expect becomes valueless [14]. Indeed, interoperability brings together the possibility of choice for consumers based on their needs and fosters competition in the industry for better products.

Part of the aforementioned inconvenience for the system's component replacement is solved by the standardization of Sensory Effects Metadata (SEM). The MPEG-V standard has played an important role in allowing different devices to render the same content since there is an engine to read and process the interoperable metadata [64]. However, the applications shall still support rendering devices from different brands that lack standardization. The issue that remains is that most solutions embed the code to handle the devices in their own application and it becomes complex for them to keep up with a gamut of ever-evolving heterogeneous devices. Moreover, sensory effects might not be the core business of the applications. Take the Windows Media Player for the sake of argument. Support for sensory effects devices could be added to it by merely creating a plugin to drive them. Irrespective of the manner in which this is achieved, it would have to cope with heterogeneity of devices, synchronization between hardware and software, interoperability between media and components, and so forth, which are not part of its core business.

This issue leads to a logical solution which decouples the processing and delivery of sensory effects from the multimedia applications. Accordingly, consumers could keep their possibility of choice and would allow multimedia applications not to lose focus on their core business. However, by introducing this kind of content over the network, new challenges are automatically imposed. Aspects such as communication between heterogeneous systems running under different operating systems, connectivity and communication issues for dealing with diversified devices from distinct brands, reliability for delivering the content from the source to the destination in real-time, the very heterogeneous requirements of mulsemedia applications under different conditions and constraints, and the quirky traits of each sensory medium type, make it a complex task of integration.

In this article, we reinforce fundamental challenges and requirements that have not been met for delivering mulsemedia to heterogeneous applications, as depicted in [50], and influenced by various system characteristics portrayed in the work of Broy [5]. To deal with them, we evolved a reusable component of the PlaySEM platform—an open distributed mulsemedia system [51]. Called PlaySEM Sensory Effects Renderer (SER), this component allowed the addition of sensory effects in diversified multimedia applications through the employment of MPEG-V. Even though it provided this facility to these applications, it did not cope with hardware heterogeneity properly since it was linked only to Arduino-based ones. Furthermore, when multimedia applications did not support UPnP—the protocol employed in PlaySEM SER's interface—they were hindered by communication issues albeit they had started being tackled in [50]. Another limitation was related to SEM; PlaySEM SER was profoundly connected to MPEG-V, and therefore, not ready for future SEM standards. From these challenges, requirements, and other shortcomings identified in the literature, we changed PlaySEM SER so that it can embrace variability at several levels in mulsemedia systems. As a result, this evolution led towards an interoperable mulsemedia framework for delivering sensory effects that we describe in this work. Figure 1 envisages the use of the evolved solution, namely PlaySEM SER 2. The results presented in this work include case studies where different profiles of PlaySEM SER 2 were set up for specific purposes.

This article is organized as follows. Section 2 presents the state-of-the-art in mulsemedia systems, focusing on technical aspects. Section 3 brings challenges and requirements that take new valences in the context of mulsemedia systems. Section 4 depicts the proposed solution, highlighting its architecture design, applied patterns, delivery, metadata processing, connectivity, interaction with timeline and event-based applications, approaches on how to extend it, and configuration strategies. Section 5 features case studies on the employment of the proposed solution. Finally, Section 6 summarizes and discusses the work, identifying topics for further investigation.

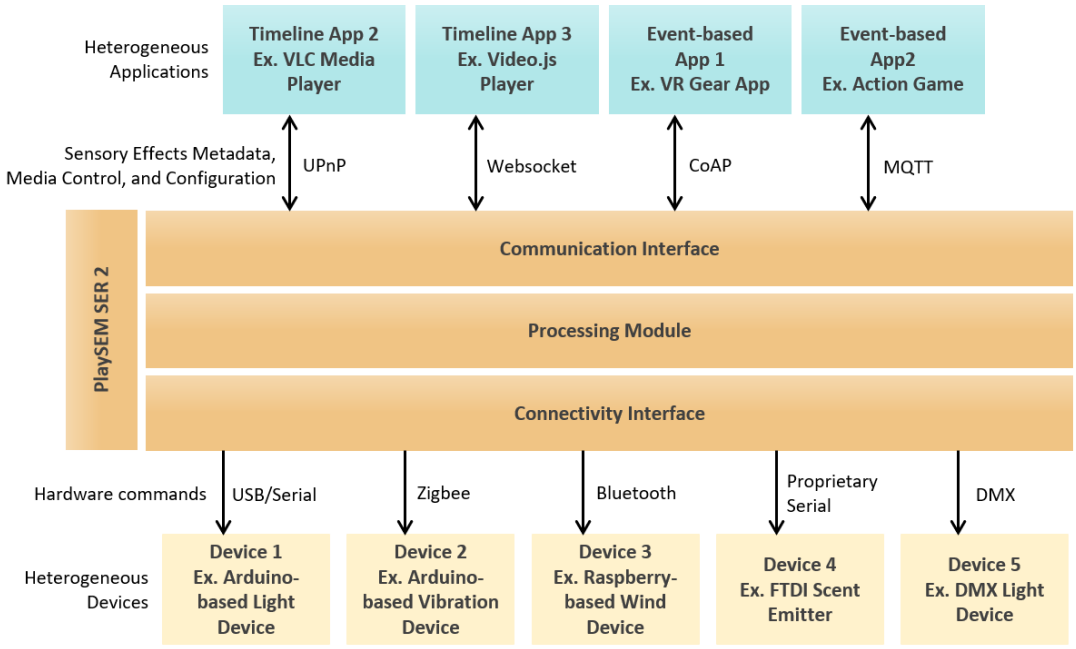

Figure 1: Envisaged use of PlaySEM SER 2 in different profiles of integration.

2 Related Work

This section encompasses comprehensive mulsemedia systems chiefly focusing on their software. Other approaches to dealing with these systems, such as conceptual architectures and frameworks, programming languages, as well as Software Development Kits (SDKs) and Application Programming Interfaces (APIs) for haptic, olfactory, and gustatory media are also covered. Devices for rending sensory effects as well as systems that cover only one sense are out of the scope of this work. We do focus on systems that use techniques to allow future extension and compatibility by using standards, protocols, design patterns, among other features, in order to deal with variability. We close this section highlighting major gaps and shortcomings to be tackled in mulsemedia systems which lead to challenges and requirements for mulsemedia systems delivery.

2.1 On Mulsemedia Systems: Brief Concepts

A distinction between multimedia and mulsemedia is made in [48]. Whilst the former aims at stimulating sight and hearing, the latter combines the former plus at least one more sense among touch, smell, and taste. The concept of mulsemedia is also slightly different from digital multisensory experiences, which do not necessarily cover multimedia. Therefore, this subtle contrast leads to different issues as mulsemedia has to work along with some audiovisual content. Another essential difference to understand the concepts placed in this realm is between application and system. There have been several meanings for these words depending on the context. In this work, we assume application as a software utilized directly by end users to perform tasks, whereas system entails a set of intertwined software and hardware that work together for a specific end, including applications.

To understand how typical mulsemedia systems work, Waltl et al. [61] presented an ordinary scenario based on a timeline multimedia application. Firstly, the main media such as a movie and its SEM file are obtained from a physical media or an online service. Then, a media processing engine acts to interpret the media resources, adapting the media, as well as the SEM, to the devices which will render the sensory effects. Also, user preferences are considered. Finally, the user environment is extended with devices (or actuators) capable of stimulating sensory effects such as vibration chairs, wind fans, scent emitters, and so on. In addition to this scenario, Santos et al. [53] envisaged mulsemedia systems working with event-based multimedia applications. Actuators can be activated by events that occur in the virtual world, such as an explosion in a game, or as a response to a stimulus from the real world captured through sensors by the virtual world. Instead of synchronizing continuous media with the actuators, event-based

multimedia applications have a requirement for quick response time, that is, as soon as an event happens the media processing engine has to deliver it as fast as it can to the actuators so as to create truly immersive experiences.

Typically, virtual information that represents sensory effects content to be reproduced in the user's environment is described through metadata. The metadata could be processed by different sensory effects engines capable of interpreting the metadata and delivering commands to specific actuators in order to be rendered. Indeed, the emerging of the MPEG-V standard to describe SEM has fostered the development of many mulsemedia software systems as shall next be described. Ideally, these systems should allow the reproduction of multimedia and mulsemedia content without customizing the solutions. However, they meet different requirements based on their needs.

2.2 Comprehensive Mulsemedia Systems

Created by Cho [9], Sensorama gives some flexibility to use SEM in timeline combined with a list of events which can be triggered on demand. On the one hand, it allows the presenter of the mulsemedia content to deliver dynamic effects in the user's environment. On the other hand, its static list of predefined sensory effects hinders its expansion. Distinct from other solutions, the Sensorama system is aimed at 4D movie theaters integrated to actuators which are part of a CAVE (Cave Automatic Virtual Environment). It supports light, wind, fog, flash-light, and vibration effects through robust devices.

SEMP[2] [60] is a player capable of reproducing multimedia content with sensory effects annotated in MPEG-V. It supports a device's amBX Gaming PC peripherals to deliver light, wind and vibration together with the Vortex Activ to disseminate aromas in the user's environment. Furthermore, it offers a poll interface so that users can give feedback about their experience. According to the results presented by its authors, users pointed out an increase in QoE when exposed to the system coupled with the aforementioned devices. Particularly, feelings of joy, fun, and worry [GG1]were heightened, whilst others such as boredom decreased. Code to deal with sensory effect is embedded in SEMP and it does not intend to deal with event-based multimedia applications. Moreover, support for additional hardware implies changing its source-code.

Based on the idea of reusing the infrastructure of sensors and actuators of a modern car, Kim and Joo [29] developed the Sensible Media Simulator. Despite being a simulator, it does not shirk away from integrating physical devices to simulate sensory effects and includes a LED lamp, a fan, a fan with a heater to simulate warmth, and a vibrator. The simulator has a web interface based on the proprietary technology Flex from Adobe which runs within web browsers. This idea comes from the purpose of allowing portability, that is, the system is able to run on different browsers. It also has independent modules for controlling different parts of the application such as presentation and processing module. However, its architecture does not allow the use of other applications as presentations. The reason for this is not to deal with communicating with devices which are not supported.

Luque et al. [33] devised the Media Processing Engine (also called Receiver Gateway). They designed and implemented a solution that integrates sensory effects into a hybrid (internet-broadcast) television system. The SEM is coded according to the format defined by the KNX (an abbreviation of Konnex) protocol, which is part of the KNX system, a bus system for building control able to exchange data via a common bus network. Data is transmitted via Wi-Fi to the receiver. The system is programmed to control a fog machine, a scent vaporizer, and an ambient light LED strip. It takes into account standardization and extensibility to some extent, but it is not designed to be reused with event-based multimedia applications. As its source-code is not available, analysis on support for new hardware is restricted.

A Multimedia-Multisensorial Platform is also presented by Bartocci et al. [4]. The idea is to create a platform to present mulsemedia by using two different strategies to transmit the media (MPEG-2 TS and IP over the network) and a hardware controller called MCU (Microcontroller Unit) to perform the conversion of sensory effects described in MPEG-V into commands for the physical devices like the Microcontroller module. However, it has some limitations, i.e. it does not allow the reuse of the MCU with other multimedia applications. The devices are capable of producing olfactory and thermal effects.

PlaySEM is a mulsemedia platform compatible with MPEG-V presented in 2015 by Saleme and Santos [51]. The platform is composed of three main decoupled components (i) the Sensory Effects (SE) Video Player[3], (ii) the SER 1.0.0[4], which processes MPEG-V metadata and prepares commands to control the devices, and (iii) an Arduino-based microcontroller module, responsible for receiving the commands and driving different actuators. This very early

[2] SEMP available at http://sourceforge.net/projects/semediaplayer/
[3] PlaySEM SE Video Player available at https://github.com/estevaosaleme/PlaySEM_SEVideoPlayer
[4] PlaySEM SER 1.0.0 available at https://github.com/estevaosaleme/PlaySEM_SERenderer/releases/tag/1.0.0

version of PlaySEM SER then evolved to its 1.1.0 version[5] to be used not only by the SE Video Player, but also by several different applications, from multimedia players to any event-based application such as games, VR/AR software, and interactive applications [46, 50]. Despite expanding its versatility, it had some limitations which shall be further explained in Section 4.3.

Jalal et al. [25] proposed an IoT-based architecture for mulsemedia delivery to TV users in a home entertainment scenario in which they used not only the aforementioned PlaySEM SER [51], but also the PlaySEM SE Video Player. The solutions were placed respectively in the Aggregation Layer and Application Layer of their IoT-based architecture, which is composed of two more layers: Virtualization, where they used another set of hardware in conjunction with PlaySEM SER and its Arduino-based microcontroller module, and Physical, where the devices are placed. This approach started making evident the reuse of PlaySEM SER albeit it had the same restrictions to deal with variability in a broader sense whereby applications and devices can be seamlessly swapped.

Another solution to deliver sensory effects for home entertainment is depicted by Lin and colleagues in [32]. Akin to the proposal of Jalal et al. [25], the authors advocate an IoT-based architecture where they combine IoTtalk Server, to control the rendering devices, with Video Service Platforms, through TheaterTalk. Applications for devices shall be created to handle them and configured on IoTtalk Server. Changeability and extendability are taken into consideration through this strategy. However, their work does not account for SEM standardization, requiring designers to annotate videos using their own authoring tool that produces content compatible with their system. This could eventually hinder compatibility between content and applications. Moreover, although it works with different video services, the use of their system with event-based multimedia applications are not addressed.

2.3 Other Proposals and Standalone Solutions

2.3.1 Conceptual Architectures and Frameworks, and Programming Languages

The works of Suk et al. [56], Choi et al. [8], and Yoon [65] are all endeavors to promote architectures and frameworks for delivering sensory effects. A device inter-locked media service framework and its technology for media generation are proposed by Suk et al. [56]. Their framework is a conceptual model where they outline the process of creating, packing, and transmitting SEM synchronizing device with media. This proposal is rather similar to the description of Waltl et al. [61] on how mulsemedia systems work. However, it does not address event-based multimedia applications and lacks practical implementations of the whole proposal.

Choi et al. [8] present concepts and guidelines for a broadcast-based framework for streaming service with sensory effects to bring 4-D entertainment for homes, relying mainly on the MPEG-V standard. However, the authors do not show the implemented system. Moreover, whereas MPEG-V introduces some standardization on their conceptual framework, it does neither consider accommodation of future standards nor event-based multimedia applications. Likewise, Yoon [65] suggests technologies to be used for delivering sensory effects in a home environment, including MPEG-2 TS (Transport Stream), MPEG-V, and UPnP. The author seems to extend what is described in [8] and first presents concerns with regard to performance, suggesting compression for SEM. Nevertheless, the limitations remain the same as in [8].

In [57], Sulema proposes a programming language for effective processing of multimodal data in order to allow the development of mulsemedia applications for several areas including education, health, among others. It resembles a declarative language to present mulsemedia content. The constraint though is that there is no real system to evaluate the development of mulsemedia systems from this programming language. Another restriction is related to support for event-based multimedia applications.

A new concept application referred to as 360° Mulsemedia envisaging a conceptual Mulsemedia Delivery System for 5G networks is introduced by Comsa et al. [10]. They propose to encode the videos embedding sensory effects into each frame, which in turn, has a matrix of intensities of scent intensities according to the user's viewport. Although they introduce a groundbreaking proposal, no standardization is used to annotate the videos and does not account for event-based multimedia applications.

2.3.2 SDKs and APIs for Haptic, Olfactory, and Gustatory Solutions of Software

Unlike mulsemedia software systems, which directly deliver multisensory experiences to the users [48], SDKs and APIs in the context of mulsemedia provide means to develop applications that support some type of sensory effect from some sort of computer application. In a nutshell, APIs offer interfaces for software to cooperate with other

[5] PlaySEM SER 1.1.0 available at https://github.com/estevaosaleme/PlaySEM_SERenderer/releases/tag/1.1.0

software whereas SDKs include a set of tools such as libraries, documentation, samples, and so forth, to enable the development of software.

Haptic. Whilst many commercial haptic devices such as Feel Three, Roto VR, Dexmo and Tesla Suit [47], to name but a few, have their own SDK to control their devices, there has been an upsurge of open-source solutions to deal with haptics over the past years. This type of interface can be used as either an input device or an output device. Even though a myriad of software solutions has been created to control haptic hardware, the scope of study in this section is concentrated on software that handles devices which generate artificial stimuli to users, that is, SDKs and APIs for tactile and kinesthetic devices.

Novint Falcon, a haptic device that offered an SDK limited to Windows [34], has contributed to foster the development of many open cross-platform plugins, libraries, SDKs, and API to control it. A specific library named libNiFalcon[6] was developed for this purpose. Many other solutions for haptics have included supported to Novint Falcon, such as JTouchTool[7], Haptik Library[8], CHAI 3D[9] [11], HAPI[10] from H3DAPI, OpenHaptics[11] [24], etc. Most of them have supported connection to other popular haptic devices including SensAble PHANToM devices, providing a layer of abstraction as well as the commercial SDK Immersion's TouchSense[12].

Javascript solutions have been also developed to provide haptic interaction for web-based applications. P4A Haptic Toolkit[13] [26], JHaptic library[14] and Haptics.js[15] are efforts to deliver vibrotactile effects providing a cross-browser compatibility layer. They do not deal with the hardware level, therefore, an SDK might be required for the former and compatibility between browser and hardware for the latter.

Other efforts to provide access to haptic devices can be found in the literature. SimHaptics[16] [52] is an open-source library to deliver haptic feedback that is compatible with devices produced by Force Dimension. To be applied in the field of training in medicine, Virtual Surgery SDK was created by Kolsanov et al. [30]. According to the authors, it provides realistic force feedback and allows the possibility to reuse its components to create other solutions. Haplet [16] provides APIs in C++ and Python to interact with computers and tablets. Both hardware and software solutions are open-source[17]. HPGE (Haptic Plugin for Game Engines) [2] was developed to provide haptics for game engines. It is based on CHAI 3D [11] and focuses extensively on Unity3D.

Olfactory. As reported by Murray et al. [38] and Howell [22], SDKs and APIs have been developed for commercial olfactory devices, such as Cyrano (for iOS devices), Scentee (for Android and iOS), Exhalia SBi4 and Dale Air Vortex Activ (Windows), and Olorama (unknown). The caveat is that these software solutions cannot be used with other devices. In other words, their business model relies on selling their hardware accompanied by software to make them work, not allowing reuse when an olfactory device is replaced. Conversely, McGookin and Escobar [35], Howell et al. [22], and Dobbelstein and colleagues [13] have created open solutions that allow some degree of reuse.

In [35], Hajukone[18] is presented as an open device for olfactory experiences. It was designed so that it could be reproducible by human-computer interaction researchers in this domain. Although it is focused on the device itself, its Arduino-based code is open as well.

The Arduino-based olfactory solution described by Howell et al. [22] is another effort to make an affordable olfactory device that can be reproduced by researchers. Akin to Hajukone [35], the work focuses mostly on the olfactory device. Likewise, there is a module written for Arduino that controls a small computer fan that delivers airflow for vaporization and scent delivery.

Dobbelstein et al. devised inScent[19] [13]—a wearable olfactory device that notifies users when receiving messages on their smartphones. The software that controls the device is also open-source, according to the authors. This device uses an Arduino-based microcontroller called BLE Nano, that allows wireless communication between the smartphone and the olfactory device. A framework is provided for smartphones to reach the device. An interface must be implemented to send messages to the Arduino module.

---

[6] libNiFalcon available at https://github.com/libnifalcon
[7] JTouchTool available at https://github.com/IanJohnArcher/JTouchToolkit
[8] Haptik Library available at http://sirslab.dii.unisi.it/haptiklibrary/
[9] CHAI 3D available at https://github.com/chai3d/chai3d
[10] HAPI (H3DAPI) available at http://www.h3dapi.org/
[11] OpenHaptics available at https://www.3dsystems.com/haptics-devices/openhaptics
[12] Immersion's TouchSense SDK available at https://www.immersion.com/technology/#touchsense-technology
[13] P4A Haptic Toolkit available at https://github.com/NickKaklanis/WebHapticModule
[14] JHaptic library available at https://github.com/guari/jhaptic
[15] Haptics.js available at http://www.hapticsjs.org/
[16] SimHaptics available at https://github.com/filipposanfilippo/SimHaptics
[17] Haptlet available at http://crgallacher.com/haply-project-open-source-haptics/
[18] Hajukone available at https://github.com/davidmcgookin/Haju
[19] inScent available at https://www.uni-ulm.de/?inscent

Gustatory. Still in its infancy due mainly to its complexity [47], there has been little work on taste for mulsemedia systems. The works of Ranasinghe and colleagues [44, 45], Karunanayaka [27], and Vi et al. [59] introduce some initiatives to deliver gustatory experiences. However, they do not provide either SDKs or APIs. An exception is an open-architecture to deliver taste synchronized with a functional magnetic resonance imaging system created by Canna et al. [7]. They developed an open software based on Arduino to start and stop the delivery of gustatory from a device they have devised. Although it has a specific purpose, their Arduino controller module named "Gustometer"[20] and device can be reused with other systems.

2.4 Gaps and Shortcomings

Over the past few years, the focus of many researchers was on understanding users QoE in mulsemedia systems [62, 43, 67, 68, 39, 37]. Whilst they have made evident increases in QoE, we predict that mass adoption of these systems will cross the line of how the experience is delivered. With constant advances in software and hardware, we believe that approaches to dealing with variability will be of paramount importance.

The works found in the literature certainly bring contributions to the mulsemedia arena. However, most of the systems presented in this section were built for specific purposes, without allowing for interoperability at the application and hardware levels, which may hinder the integration with other applications in home environments, 4D movie theaters, VR applications augmented with multisensory effects, etc., and hardware for delivering sensory effects. Conversely, some authors present proposals of conceptual architectures and frameworks taking into account some degree of standardization but these ideas still need to be evolved and materialized somehow to real environments. As proposed by some authors, the IoT might emerge as a solution to integrate different devices in mulsemedia environments. However, if not properly assembled, the system can become cumbersome and as a result, include undesired delays. Furthermore, IoT solutions would have to adapt themselves to support interaction with heterogeneous timeline and event-based applications and take into consideration standardization and techniques to cope with SEM.

As for SDKs and APIs, they have been developed to deal with each sense separately. They usually aim at working as interfaces for specific devices. Despite this limitation, they can be combined in a framework to perform varied tasks. Haptic has appeared as the most supplied of software solutions to cope with heterogeneity, whereas olfactory and gustatory have been little explored due mainly to the shortage of off-the-shelf devices.

Although there have been initiatives approaching software and hardware heterogeneity, there are still improvements to be made so that mulsemedia solutions become polyvalent, reusable, and mutable without being detrimental to users' QoE. By detrimental we mean that those craved characteristics do not interfere mainly on temporal aspects. Tolerable delays for multisensory interactions have been identified in the literature. These times are quite strenuous to pinpoint once they might vary from one context to another. For instance, Nakamura and Miyashita [40] found out that electric taste can be presented with visual stimuli with a difference between [13ms, 837ms]. Murray et al. [39] discovered different thresholds depending on the scent type, that is, foul scent [0s, +15s], spicy, fruity, flowery scents [-10s, 10s], and burning scents [-5s, +10s]. For haptic accompanied by videos, Yuan et al. [66] indicated [0s, 1s] as a tolerable range. On the other hand, when haptic is presented with a head-mounted display, like the work of Adelstein, Lee, and Ellis [1], system latency must be no higher than 17ms. As pointed out by Kim, Osgouei, and Choi [28], users negatively notice even a 40ms delay in a touchscreen event-based application with haptic feedback. Regardless of the particularity of each case, it is clear that there is a concern about temporal aspects in mulsemedia systems that shall be taken into account.

Mulsemedia systems give rise to a complex scenario stemming from their unconventional needs while producing, transmitting, integrating, and presenting distinct sensory effects under varied constraints and conditions. Therefore, variability is one of their noticeable traits. From the perspective of current work, we have listed the following major gaps and shortcomings to be evenly dealt with in mulsemedia systems:

- Mulsemedia renderers have been coupled in multimedia applications making reuse with other applications harder;
- When mulsemedia renderers are decoupled from multimedia applications, they lack support for varied protocols to interact with them;
- When mulsemedia solutions are able to work with timeline-based multimedia applications, they seldom give support for event-based ones;
- Whilst MPEG-V has been supported by some mulsemedia systems, they are not prepared for changes in the

[20] Gustometer available at https://github.com/antocanna88/gustometer

event that they need to support SEM in another standard;
- Delay is rarely taken into account—mulsemedia systems lack mechanisms to eventually compensate delays introduced for some hardware or connectivity and communication protocols;
- Hardware heterogeneity is mostly tackled isolated by some SDKs or APIs (in haptic) which means that mulsemedia systems hardly ever provide configurable means for device replacement;
- Mulsemedia systems lack some way to adapt themselves for different profiles of usage without changing their code;
- Noticeably, most mulsemedia systems do not consider future growth and rely only on existent technologies, protocols, and standards.

In the next section, we discuss the challenges and requirements that take totally new valences for mulsemedia systems taking into account these major gaps and shortcomings.

3 Challenges and Requirements for Mulsemedia Systems Delivery

From the related work section, it is not hard to realize the multitude of concerns that mulsemedia systems have to cope with. Software-based systems are increasingly become bigger and more complex due to different scenarios, conditions and constraints they have to operate with. Mulsemedia systems fit this context. Major obstacles and issues in deploying mulsemedia systems have been identified in [19, 38, 12, 49[GG2]]. They include mulsemedia integration, synchronization, intensities, wearable and other heterogeneous devices for delivering sensory effects, and remote delivery of mulsemedia components [48]. Thereby, focused chiefly on the design and implementation of mulsemedia systems, we reinforce fundamental challenges for delivering multisensory effects to heterogeneous applications first described in [49] under the influence of Broy [5].

Sections 3.1 to 3.3 characterize the identified challenges and establish requirements to be addressed in order to face each of them. These requirements are identified by an ID (C_iR_j stands for Challenge i, Requirement j) that relates the challenges to requirements to be referred to subsequently in the text.

3.1 Challenge I - Multifunctionality and Reusability

Because of the frequent advances in mulsemedia devices, mulsemedia systems should be able to accommodate these changes constantly. At the same time, new approaches to interacting with multimedia applications have emerged such as multi-touch interfaces, voice processing, and brain-computer interfaces, giving rise to new kinds of complex interactive systems. As a result, mulsemedia systems have to be able to adapt themselves to different circumstances of operating systems, presentation interfaces, standards, connectivity and communication with applications and devices. Due to this continuous evolution, reusability is key to dealing with heterogeneous settings.

In cases where multimedia applications, such as video players, games, VR-360 applications, have to be integrated with new sensory effects devices taking into account different operating systems, requirements for mulsemedia systems to be multifunctional (*C1R1*), compatible (*C1R2*), portable (*C1R3*), and compliant/interoperable (*C1R4*) should be met. *Multifunctional* means that a mulsemedia system will provide operations for multimedia applications to add multisensory effects into it, being able to work with timeline and event-based multimedia applications. It also has to be *compatible* with its counterpart, i.e. it should provide reusable implementations of services to communicate with the multimedia applications. Moreover, it should be able to run in the expected environments that the user needs (*portable*). This leads to an explicit requirement for compliance, that is, mulsemedia systems should meet rules or standards to allow interoperability in several levels.

Architectural patterns should be applied to mulsemedia systems for allowing reuse. By detaching layers, multimedia applications could use common services from an interoperable multimedia system that decouples presentation from mulsemedia processing and rendering. For instance, it[GG3] could keep apart concerns not directly related to mulsemedia systems such as adaptive streaming, buffering, video encoding and decoding, among others.

3.2 Challenge II - Reactivity and Timeliness

Event-based multimedia applications, such as games, VR/AR software, and interactive applications, are ones for which every second is precious. Mulsemedia systems, in this case, have to work in real-time reacting as soon as

possible to events generated by event-based multimedia applications swiftly controlling all the sensory effects devices required to perform an action. Not meeting real-time deadlines would be detrimental to user QoE. Therefore, this challenge mandates that mulsemedia systems be reliable (*C2R1*), and responsive (*C2R2*).

Responsiveness in real-time event-based mulsemedia applications is crucial. An input will entail one or more reactions in mulsemedia systems instantaneously. It shall be *reliable* in order to perform actions consistently producing correct outputs under a time constraint. To be *responsive* signifies that a short response time is required for some situations such as in a game where an explosion occurs and wind and vibration devices cannot deliver late effects. Thus, delay is an issue that needs to be taken into account in mulsemedia systems mainly in application level where most heavy operations are processed.

3.3 Challenge III - Manageability and Configurability

Complex architectures composed of heterogeneous devices present a significant challenge for mulsemedia systems. A device can be connected to a mulsemedia system through many connectivity protocols such as Wi-Fi, Bluetooth, Zigbee, and so on, allowing wireless communication. In addition to that, a multimedia application could impose different communication protocols on the mulsemedia system.

A scenario whereby a scent device replacement is necessary can exemplify this. In the event that a new device has novel features but uses a different connectivity protocol, this change becomes a hurdle. Considering that the former connectivity was hard-coded in the system, that is, the services provided by the mulsemedia system were highly dependent on it, this can become even worse once low cohesion and high coupling are unbidden guests at the developer's dinner table.

Therefore, mulsemedia systems should be able to work with heterogeneous multimedia applications without difficulty. This is the case in which architectural and design patterns boost mulsemedia systems to be built so that they are *changeable* (*C3R1*), *extensible* (*C3R2*), *orthogonal* (*C3R3*), and *adaptable* (*C3R4*). The ability to be *changeable* features mutability in its structure without interfering with the adjacent layers of the mulsemedia system. Extensibility plays an important role in changeability in the sense that the design of the system takes future growth into consideration. Orthogonality in architectural level will also aid the designer to anticipate possibilities to combine different components. Adaptability makes available mechanisms to customize the system to their needs without needing to change the system, that is, mulsemedia systems should be designed so that users can configure and extend the application according to their specific needs.

Figure 2 portrays a summary of the major challenges and requirements for mulsemedia systems delivery described in this section.

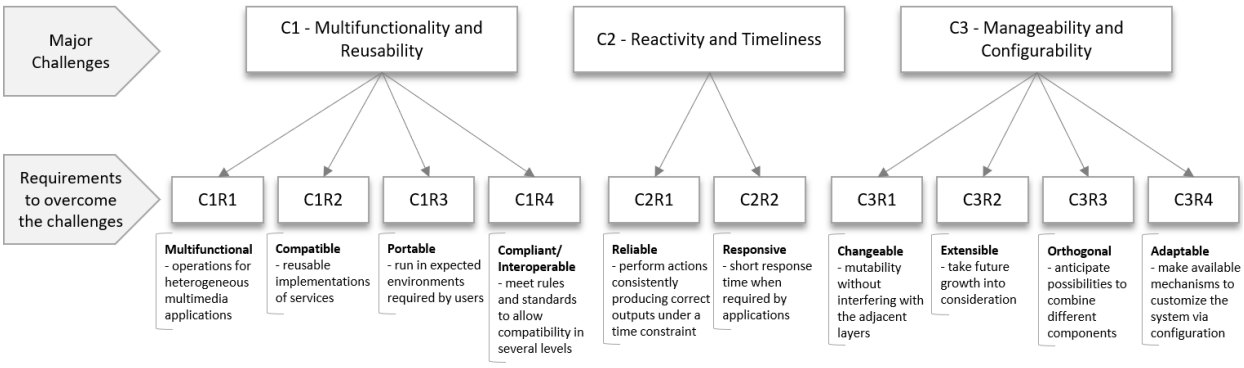

Figure 2: Summary of the major challenges and requirements for mulsemedia systems delivery.

4 Delivering Sensory Effects to Heterogeneous Systems

The evolution of PlaySEM SER was designed to tackle the challenges previously discussed. In this section, we present in detail the characteristics of the proposed framework, PlaySEM SER 2[21], for working with heterogeneous

[21] PlaySEM SER 2.0.0 available at https://github.com/estevaosaleme/PlaySEM_SERenderer/releases/tag/2.0.0

applications and devices linking the particulars to the requirements. We show how the framework is capable of integrating any application by configuring and taking advantage of its flexible and scalable architecture to support new communication protocols, standards, connectivity, and devices.

4.1 Architectural Design

Software architecture representations provide a holistic view from abstractions, which represent the structure and organization of the components as well as their relationships [42]. The ISO/IEC 42010:2011 [23] standard describes software and systems architecture as fundamental concepts and properties of a system and its environment, personified in its elements, relationships, and principles of its design and evolution. The architectural aspect is the first stage in the software design process [55]. It helps to identify structural components. As an output, the design phase should describe how the software is organized through a set of components which communicates with each other.

The architectural design of a system provides agility to disseminate knowledge about its structure. In this fashion, Figure 3 depicts the elements of the proposed architecture in a bottom-up approach whereby *Multimedia Applications* are beneath mulsemedia features. It is represented in a block diagram, which is commonly used for representing systems architecture as it presents a system high-level view that allows people from different disciplines to understand it [55]. On top of *Multimedia Applications* are blocks corresponding to *Communication Broker*, *Sensory Effects Processing*, *Connectivity Interface*, and *Rendering Devices*. This layered division supports a horizontal extension to accommodate new technologies without changing adjacent layers.

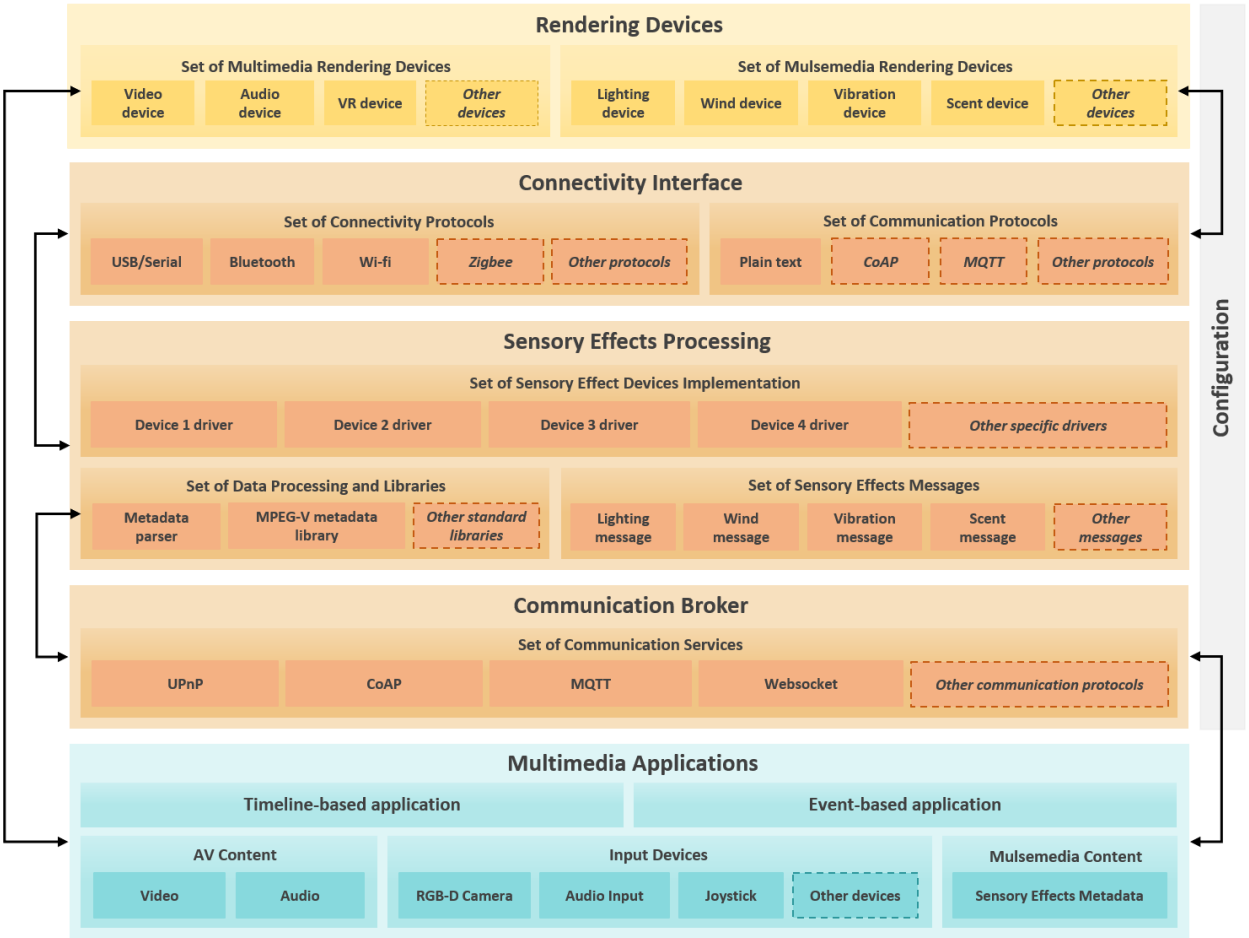

Figure 3: Architectural design of PlaySEM SER 2 and other mulsemedia system elements from a bottom-up perspective.

The *Multimedia Applications* block encompasses several components linked to applications whereby the user interacts actively. The *Timeline-based application* is, as its name suggests, one that is based on a representation of a

period of time such as a video or audio player. The *Event-based application* box represents those categories of applications in which actions are triggered as a result of events that occur in the virtual world, such as an explosion in a game, or as a response to a stimulus from the real world captured through sensors by the virtual world. Both *Timeline-based application* and *Event-based application* can make use of *AV Content* such as *Video* and *Audio*. Furthermore, *Input Devices* such as an *RGB-D Camera*, an *Audio Input*, or a *Joystick*, can be provided to the applications as a mean of capturing the users' interaction. Finally, a *Mulsemedia Content* could also be provided in this layer as *Sensory Effects Metadata* associated with *AV Content* or actions triggered in event-based applications.

Communication Broker, *Sensory Effects Processing*, and *Connectivity Interface* form PlaySEM SER 2's core. The *Communication Broker* works as a facade for *Multimedia Applications* to convey the SEM to PlaySEM SER 2 and to provide reusable abstract services for starting, pausing, stopping, and synchronizing multimedia content/interaction with sensory effects devices. This facade provides different ways to interact with *Multimedia Applications*—it works according to the application's category. The communication between *Multimedia Applications* and PlaySEM SER 2 *Communication Broker* can be established through many protocols such as *UPnP*, *CoAP*, *MQTT*, and *WebSocket*. *Other communication protocols* can be easily provided in this layer. The protocol to be used will depend on the application's requirement. Details on how to interact with the services provided by the *Communication Broker* and add new protocols are described in Section 4.4 and 4.7.

After establishing communication with PlaySEM SER 2 and conveying the SEM, it is time for the *Sensory Effects Processing* to interpret and convert metadata into messages to control the devices. The messages are non-device-dependent which means that the *Metadata parser* is able to queue generic messages related to *Lighting message*, *Wind message*, *Vibration message*, and *Scent message* to subsequently deliver them to the specific devices configured in PlaySEM SER 2. *Other messages* types can be added to support other kinds of sensory effects. This isolation level gives flexibility for mulsemedia systems to work with any standard and any device. Natively, PlaySEM SER 2 supports the MPEG-V standard through the aid of the *MPEG-V metadata library*. *Other standard libraries* can be added to deal with future standards. Also, in this layer, specific device drivers are implemented. Though most devices require a specific connectivity protocol, at this point no connectivity is provided, which means that the same device can be connected by using different protocols. The implementation of the drivers is concerned with how to convert the queue teemed with generic messages into specific commands for the devices. Furthermore, they can make a bridge between PlaySEM SER 2 and SDKs and APIs by implementing means of interfacing with them. Thereby, PlaySEM SER 2 is able to embrace as many devices as it can from different brands and with distinct particularities.

The *Connectivity Interface* allows PlaySEM SER 2 to establish connections with multiple and heterogeneous devices. A set of protocols such as *USB/Serial*, *Bluetooth*, and *Wi-Fi* are supported. *Zigbee* is suggested due to the fact that this protocol is used in many portable devices. However, it does not hinder the expansion of *Other protocols* such as Sigfox and 6LowPAN, for instance. To exchange messages between the devices and PlaySEM SER 2's *Connectivity Interface*, communication protocols such as *CoAP* and *MQTT* are suggested when needing a standard, but *Plain text* represented in bytes is commonly used to convey the commands to the devices. *Other protocols* are permitted as well.

The *Rendering Devices* layer separates the output devices that are responsible for conveying content from the virtual to the real world. *Video device*, *Audio device*, *RV device*, among The *Other devices* container, represent devices to present audiovisual content for *Multimedia Applications* whereas *Lighting device*, *Wind device*, *Vibration device*, *Scent device*, among *Other devices*, represent sensory effects devices connected to PlaySEM SER 2 for providing multiple sensory effects in *Multimedia Applications*.

Finally, the *Configuration* cross-layer aids on customizing PlaySEM SER 2 to be executed under different compositions and setup.

4.2 Design Patterns Applied

A layered architecture subdivides solutions so that each tier can evolve separately. Criteria for separating concerns may vary from system to system and include abstraction, granularity, hardware distance, and rate of change [6]. Concurrently, design patterns aid developers to make design choices so that they do not compromise reusability [18]. Patterns allow reusing the experience of others who experienced similar problems by describing a recurrent problem in an environment and a core solution without ever doing it the same way twice.

Within PlaySEM SER 2, we have applied the patterns *abstract factory*, *singleton*, *facade*, *observer*, *strategy*, and *template method* to make the solution more flexible and ultimately reusable. These patterns help to avoid many problems such as creating an object by specifying a class explicitly, dependence on hardware and software platform, dependence on object representations or implementations, tight coupling, and extending functionality by subclassing.

The pattern *abstract factory* was used to provide interfaces for devices, services, connectivity, and messages without specifying their concrete classes. An instance of a concrete class, which contains specific implementations, is created at run-time in this case. The main idea is that the system becomes independent of how its products and services are created, composed, and represented, therefore, promoting consistency among the classes and isolating specific solutions.

An application needs only one instance of a factory per group of matter which leads to the use of the pattern *singleton*. It provides a central point of access for services, devices, and connectivities within PlaySEM SER 2. Moreover, the system relies on this pattern for accessing a single instance of a timeline per event so that the sensory effects are arranged over it.

Abstract factory classes are often implemented with *factory methods*, which in turn, let subclasses redefine certain steps of an algorithm without changing the algorithm's structure. It is useful to implement the invariant parts of an algorithm leaving it up to subclasses to implement the specific behavior. In the following sections of this article, abstract services, devices and connectivities classes implementing the invariant steps of the algorithm will be presented. As distinct to *factory methods*, the pattern *strategy* was applied to vary entire algorithms eliminating conditional statements. It is a choice of implementation. For instance, this is the case where a new device for a specific kind of effect should be selected instead of another or a new SEM standard emerges. It is used mainly for the communication broker, metadata parser, devices, and connectivity interfaces within PlaySEM SER 2.

To provide a high-level of abstraction for PlaySEM SER 2 and expose elementary mulsemedia services to multimedia applications, we applied the *facade* pattern. It helps to layer the system, i.e. the communication broker is a *facade* for multimedia applications whereas connectivity interface is a *facade* for the drivers of the devices providing operations to open and close connections, besides sending a command to a device. Therefore, the communication between the layers is simplified through *facades*. It also promotes weak coupling between systems. Generally, *facades* require only one instance of an object having a straight relationship with *singletons*. In addition, *facade* is combined with *abstract factories* within PlaySEM SER 2 so that the behavior of a service obeys what is defined through a selected strategy.

Last but not least, the pattern *observer* (also know as publish-subscribe) was applied to PlaySEM SER 2. After processing the SEM, the system should notify the integrated multimedia application (subscriber) at the end of this process so that other operations to control the devices can be started. In the case of the classes of services provided by the communication broker, the abstract service class fires a property change so that the concrete classes reach the multimedia application via their own standards of communication.

4.3 PlaySEM SER's Evolution

PlaySEM SER has evolved both vertically and horizontally to meet the aforementioned requirements, to accommodate changes, and thus, to become a framework. Vertically leads to the separation of levels to cope with abstraction whereas horizontally means having logical components to deal separately with different concerns. Previously in [51], PlaySEM SER did not have an adaptable and scalable architecture whereby new and forthcoming protocols and technologies could be included without changing its components despite being decoupled from its multimedia counterpart. A device replacement meant it should be compatible with its same microcontroller and the code of the latter updated. Moreover, it was inextricably linked to MPEG-V which hindered the use of PlaySEM SER with either other standards or someone's own metadata. It did not provide support for smell and dealt inadequately with event-based applications. Third-party components such as libraries that implement communication and connectivity protocols were not managed, running the risk of being out-of-date. A change meant an alteration in the code, not only in a configuration file. As reuse is a pivotal aspect of PlaySEM SER 2, Java *Generic Types* with *Reflection* were combined to support a set of architectural and design patterns that allow it to be applied to different contexts. Therefore, it went from 11 classes (original PlaySEM SER) to 48 classes (PlaySEM SER 2) as available in its GitHub repository. This significant changes made possible the introduction of these new features. Particulars of them are described next.

Since its first version described in [51], PlaySEM SER has been developed in Java. Reasons for this choice stem chiefly from the following facts: (i) Java is rather popular—it is second most tagged programming language on GitHub (a code hosting platform with more than 2 million active repositories) up to the date of this paper [20], just behind JavaScript; (ii) Java is platform independent, which means that software developed in this language is able to run on different types of machines. This would be especially useful, for instance, to port PlaySEM SER to trendy wearable mulsemedia systems; (iii) it is object-oriented (OO), which means that a plethora of OO widespread design patterns can be applied to PlaySEM SER; (iv) it is less complex than C++; and (v) there are several third-party

resources that are developed as libraries that can easily be integrated into PlaySEM SER to expand support for new protocols for either connectivity or communication, for the sake of argument. There are other languages that fulfill some of these reasons. Thus, as stated in [49], other programming languages can be used to adapt this framework following the model proposed by the authors.

4.4 Communication Broker Implementation

PlaySEM SER 2's communication broker has the purpose of making easier the reuse of services provided by the SER via a single point of access for multimedia applications under the operation of many protocols, according to the need of the application. The SER currently supports the UPnP, CoAP, MQTT, and WebSocket protocols, but its design allows the inclusion of others without changing or implementing its services.

UPnP defines an architecture for communication in pervasive networks [58]. The proposal is that devices and systems from different brands communicate with each other based on the HTTP protocol and XML language, offering flexibility and ease of use. It provides automatic discovery, description, control, presentation and sending of events remotely.

CoAP is a transfer protocol designed for use with constrained environments, that is, systems with restricted conditions [54]. CoAP is based upon Representational State Transfer (REST) architecture providing a request/response and publish/subscribe interaction with very low overhead. It runs on UDP, however, it has its own reliability mechanism when a message confirmation is needed. According to Shelby et al. [54], it could also be used over other transport protocols such as SMS, TCP, or SCTP. Kovatsch et al. [31] reported that their framework, entitled as Californium CoAP, showed 33 to 64 times higher throughput than high-performance HTTP Web servers. Thus, CoAP has a promising performance in distributed systems.

Just like CoAP, MQTT is a lightweight protocol thought to work with resource-constrained devices [3]. However, it is limited to a publish/subscribe interaction based on topics and runs on top of TCP/IP. The protocol itself is designed to be simple. A client can publish messages to the broker and subscribe to topics for receiving events. The protocol has three qualities of service for message delivery. These are: *At most once*, with a low guarantee, in which message loss can occur; *At least once*, with guaranteed delivery but duplications can happen; and *Exactly once*, in which has the assurance that a message will arrive just once. The intrinsic characteristic of MQTT makes it a minimized protocol to reduce network overhead.

Unlike CoAP and MQTT, the WebSocket protocol was not designed originally for constrained devices. It aims to provide a mechanism for browser-based applications, such as games, simultaneous editing tools, user interfaces exposing server-side services in real-time, and so on, that need two-way communication with servers that do not rely on opening multiple HTTP connections [15]. In contrast to polling, it does not have to repeat HTTP headers in each request, attenuating communication overhead. It helps developers to build scalable real-time web applications through a single socket over the internet via a browser [41]. The fact that many applications nowadays are implemented in browsers, and most of them provide support for WebSockets, makes it a requirement for PlaySEM SER 2 to incorporate.

The services provided by PlaySEM SER 2 aid the functionalities of the system from the point of view of multimedia applications. Table 1 presents a description of these services and their prerequisites. An identifier containing the prefixes *E* and *T* is given for representing operations related to Event-based and Timeline multimedia applications.

Additional services and other protocols can be added at this layer. Figure 4 shows an excerpt of PlaySEM SER 2's class diagram for services. There is an interface *SEService* that determines which services will be implemented. The abstract class *SEServiceBase* defines the behavior of each service, except for the operation *firePropertyChange* which must be implemented in each individual specialization of the *SEServiceBase* abstract class. The concrete classes *SERendererCoapService*, *SERendererWebSocketService*, *SERendererUPnPService*, and *SERendererMqttService* allow the exposure of the implemented services respectively through the protocols CoAP, WebSocket, UPnP, MQTT. Owing to a payload limitation in CoAP, there is a directional dependency from *SERendererCoapService* to *SERendererWebSocketService* so that the metadata is received through WebSocket to overcome this issue. The other services operate without restriction when using CoAP.

To add support for future protocols, a new class is included extending the *SEServiceBase* abstract class and described in the configuration file (Section 4.8). In the case of new common services, they should be implemented in the abstract class so that their behavior is propagated to subclasses not affecting the structure of PlaySEM SER 2. The incorporation of varied communication protocols allows multimedia applications to choose the best interface for them not only based on performance, but also on their own requirements and constraints. For instance, manipulating

WebSockets on web applications would require less effort than UPnP.

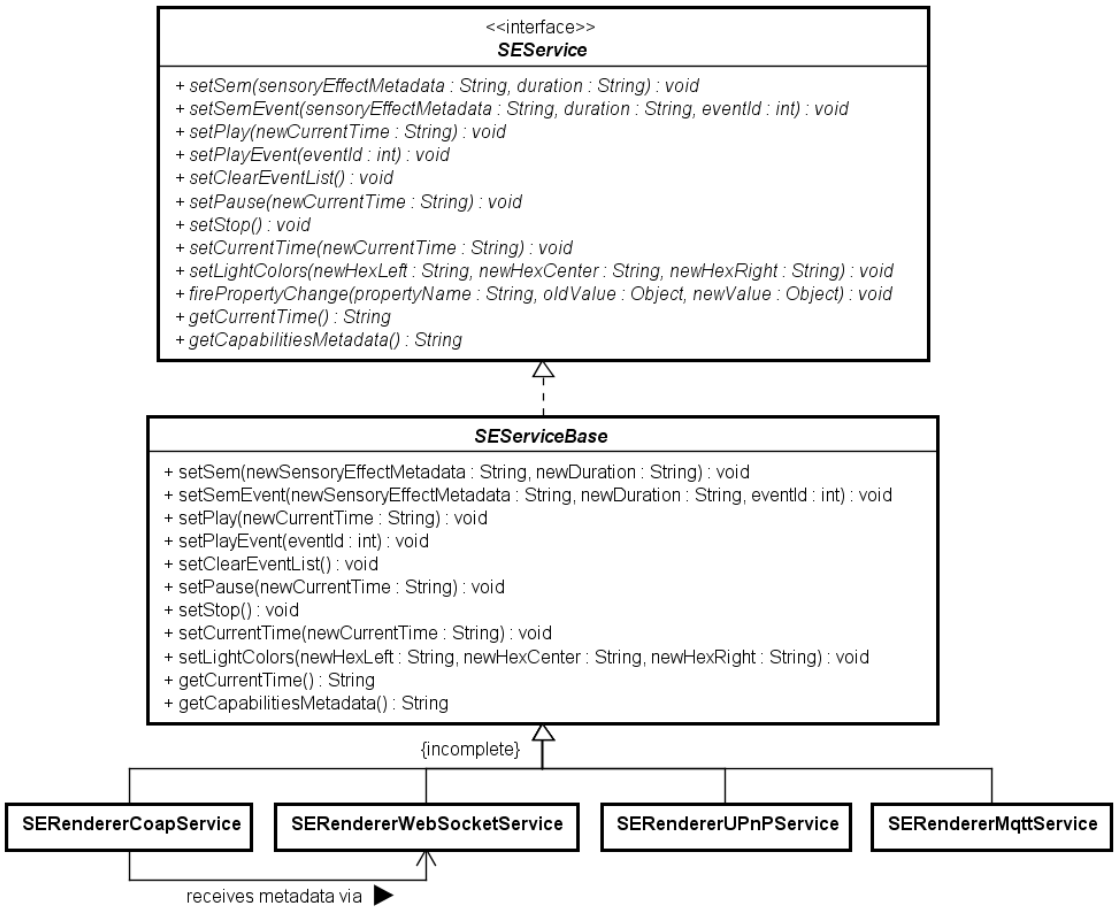

Figure 4: Excerpt of the PlaySEM SER 2's class diagram for services. The generalization is *incomplete*—other classes of services can be added.

4.5 Processing, Standardization and Devices' Implementation

Metadata processing converts a metadata script into commands for the sensory effect devices. This processing is invoked by the services *ES01* and *TS01* (see Table 1). Basically, they work in a similar way. What differentiates one from the other is that the former has a set of queues, one for each event. In a nutshell, multimedia applications provide a SEM as an input to PlaySEM SER 2, which in turn, read each element in order to give appropriate commands for different devices. Finally, the multimedia application is notified about the finishing of the processing.

The fundamental aspect of this process is that it does not know which device will receive the commands. PlaySEM SER 2 dynamically allocates the devices that will be used in an instance through configuration including its properties such as delay time and mode of connectivity. At run-time, the system retrieves these configurations and formats specific messages to the selected devices. Another key aspect is related to the strategy for parsing the SEM. A specialized algorithm is created for each standard keeping the same generic operations. In this case, there is no need to implement nor extend classes. Furthermore, types of effects not supported by the SER can be included in the metadata parsers. Before running PlaySEM SER 2, the parser must be indicated as well (see Section 4.8). The abstraction at this level assists in separating the concerns about processing, standards, and devices.

Regarding the implementation of the specific devices, it is akin to the scheme used for services. Figure 5 presents an excerpt of PlaySEM SER 2's class diagram for devices.

Table 1: Services provided by PlaySEM SER 2 to promote integration for heterogeneous multimedia applications.

Id	Service	Description	Prerequisite
ES01	SetSemEvent (metadata, duration, eventId)	It processes sensory effects metadata, converting it into specific commands for handling the devices for a specific duration and a single event.	-
ES02	SetPlayEvent (eventId)	It starts rendering sensory effects for a specific preprocessed event.	ES01
ES03	SetClearEventList ()	It empties a list of previously preprocessed events.	ES01
TS01	SetSem (metadata, duration)	It processes sensory effects metadata, converting it into specific commands for handling the devices for a specific duration.	TS01
TS02	SetPlay (time)	It starts the rendering of sensory effects at a specific time.	TS01
TS03	SetPause (time)	It freezes the rendering of sensory effects at a specific time.	TS01, TS02
TS04	SetStop ()	It stops the rendering of sensory effects.	TS01, TS02
TS05	SetCurrentTime (time)	It places the cursor of execution at a specific time.	TS01
TS06	SetLightColors (leftColor, centerColor, rightColor)	It adjusts the colors of the lighting device in real-time. This service is used for a specific purpose of transmitting the colors extracted from a frame of a video to the device. It is separated into 3 parts which are left, center, and right of the device.	-
TS07	GetCurrentTime ()	It returns the current time on the SER. It is useful for synchronization purposes.	TS01, TS02
TS08/ES04	GetCapabilitiesMetadata ()	It returns the capabilities of the devices in a standard.	-

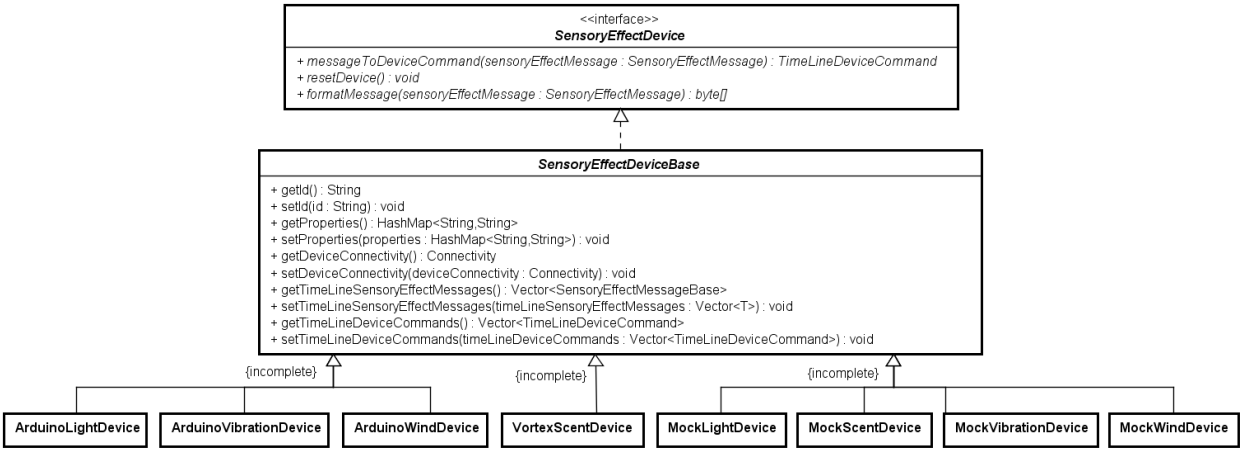

Figure 5: Excerpt of the PlaySEM SER 2's class diagram for devices. Once the generalization is *incomplete*, other classes of devices can take part.

The interface *SensoryEffectDevice* rules which operation will be implemented in all specific devices. The abstract class *SensoryEffectDeviceBase* provides the implementation of a set of common operations for each device. These operations are mainly concerned with aspects such as converting patterns of messages, retrieving specific properties for each device and linking them to predefined modes of connectivity. The concrete classes *ArduinoLightDevice*, *ArduinoVibrationDevice*, *ArduinoWindDevice*, and *VortexScentDevice* have a particular implementation considering specific outputs. The classes *MockLightDevice*, *MockVibrationDevice*, *MockWindDevice*, and *MockScentDevice* are predefined to use the console to output the outcome of the processing for the sake of simulation (see Section 4.9).

To expand PlaySEM SER 2 to support other devices, a new specific device class should be created extending the *SensoryEffectDeviceBase* abstract class bearing in mind that the operation specified in the interface *SensoryEffectDevice* must be coded. A description of the added device must be specified in the configuration file (Section 4.8). A particular feature allows developers not to be concerned about connectivity and focus only on the commands to control the new device. Through configuration, it is possible to reuse implemented connectivities as presented in Section 4.6. In the event that a device cannot avoid the use of either SDKs or APIs, an extension of *SensoryEffectDeviceBase* can be created to interface with them. For this specific case, connectivity interfaces to access SDKs or APIs should be also be created.

4.6 Connectivity with Devices

Devices are constantly changing and sometimes a replacement is needed when a user wants a new feature. For example, instead of a wired connectivity one may want to use wireless connectivity when, for instance, wires are unwieldy. That could be the case of migrating to another pattern of connectivity without changing the behavior of the driver. PlaySEM SER 2 takes this issue into account by decoupling the connectivity layer from the drivers for commanding the devices. Section 4.8 indicates how to switch protocols without changing code.

Figure 6 presents an excerpt of PlaySEM SER 2's class diagram for device connectivities in order to help understanding how to include new connectivity protocols. The interface *Connectivity* plays a role in the shape of a contract that the concrete classes have to sign. Basically, there are three operations to handle a connection, which include opening, closing and sending a message through the supported protocol. The abstract class *ConnectivityBase* implements some common operations for the connectivities such as knowing if the connection has been established and getting specific properties for each connectivity, for instance, a virtual port the protocol should use to communicate. The concrete classes *EthernetWifiOutput*, *BluetoothOutput*, *ConsoleOutput*, *FTD2XXOutput*, and *SerialOutput* implement specific directives to establish connections with Ethernet and Wi-Fi, Bluetooth, and USB/Serial devices, proprietaries (such as FTD2XX) or not. The *ConsoleOutput* helps the mock classes described in Section 4.5 to simulate devices delivering messages on the console on which the system runs.

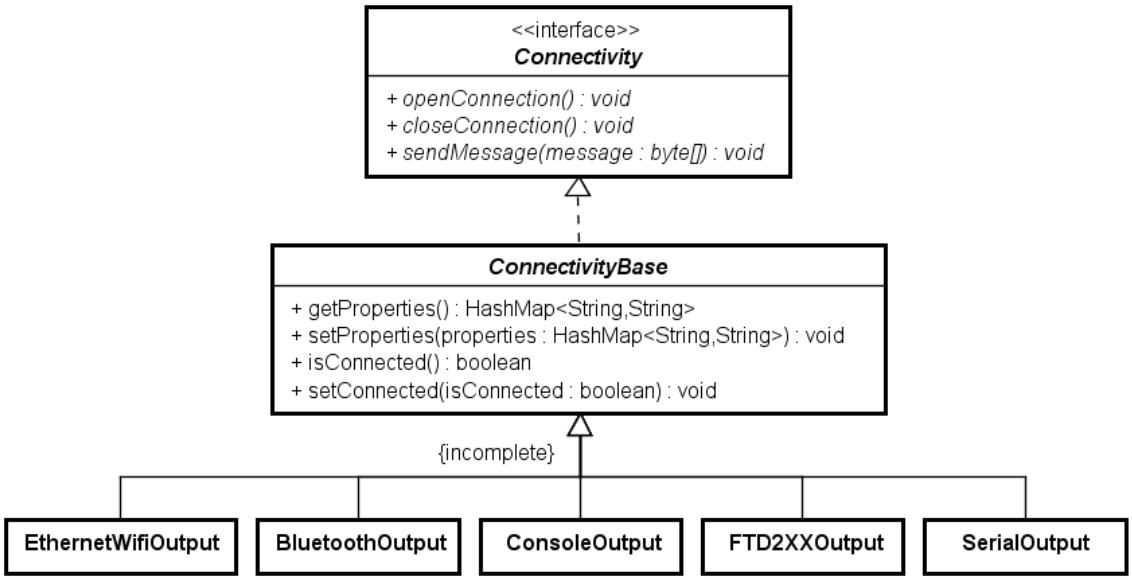

Figure 6: Excerpt of the PlaySEM SER 2's class diagram for device connectives. The generalization is *incomplete*, meaning that other connectivity classes can be considered.

Like the other extensions for services and devices, to include other connectivity protocols it is necessary to extend the abstract class. A new class should be added extending the class *ConnectivityBase*. The operations of the interface *Connectivity* shall be coded. The same pattern should be used for either SDKs or APIs as though they were connectivity means to access the devices. Moreover, the new connectivity has to be described in the configuration file (Section 4.8).

4.7 PlaySEM SER Interaction

PlaySEM SER 2 is designed so that it is easy to be integrated with timeline and event-base multimedia applications, as depicted in Section 4.1. This section describes the collaboration that must be woven between multimedia applications and PlaySEM SER 2 by using the services provided by the latter (see Table 1). The focus is predominantly on the relationships between each other.

4.7.1 Timeline Applications

The devices responsible for rendering sensory effects are synchronized with a continuous medium in the virtual world, i.e. movies, songs, and so on. As soon as a medium is open, the SEM file should be sent to PlaySEM SER 2. Additionally, at every key moment in which the user is interacting with the interface such as playing, pausing, stopping, forwarding or rewinding a medium, the same information must be communicated to the mulsemedia system. Figure 7 outlines this process of collaboration. After initializing the *Timeline App*, the *Timeline App Controller* conveys the SEM to the *PlaySEM SER Communication Broker* through the service *SetSem*. After that, the *PlaySEM SER Controller* parsers the metadata converting it into specific commands for controlling the devices and notifies the *PlaySEM SER Communication Broker* to relay this information to the *Timeline App Controller*. Following that, the service *SetPlay* should be transmitted from the *Timeline App Controller* to the *PlaySEM SER Communication Broker* to notify the *PlaySEM SER Controller* that the sensory effects can be rendered. Other services could be called here always obeying the prerequisites described in Table 1.

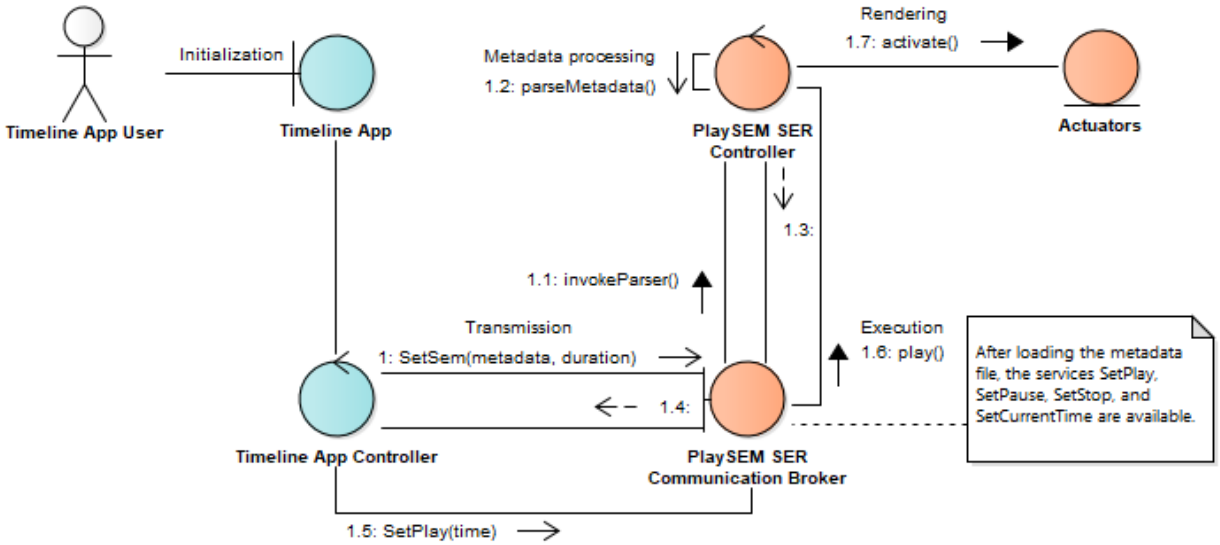

Figure 7: Interaction between timeline multimedia applications and PlaySEM SER 2.

4.7.2 Event-based Applications

The devices are usually activated by events that occur in the virtual world, such as an explosion in a game, or as a response to a stimulus from the real world captured through sensors by the virtual world. Contrary to the interaction for timeline applications, the processes for event-based applications encompasses two steps. The goal is to reduce repetitive tasks and the number of exchanged messages through the network [50]. Figure 8, item (a), presents the first step. As soon as the *Event-based App Controller* and the *PlaySEM SER Communication Broker* establish a connection, the metadata for each event should be transmitted in a loop through the service *SetSemEvent* before the user starts interacting with the multimedia application. Next, the *PlaySEM SER Controller* parses the metadata converting it into specific commands for controlling the devices, stores it in a queue identified by the id of the event and notifies the *PlaySEM SER Communication Broker* to relay this information to the *Event-based App Controller*. In the second step (Figure 8, item b), as soon as the *Event-based App* receives a stimulus, the service *SetPlayEvent* is conveyed to the *PlaySEM SER Communication Broker*, which in turn, seeks the event id and promptly triggers the devices.

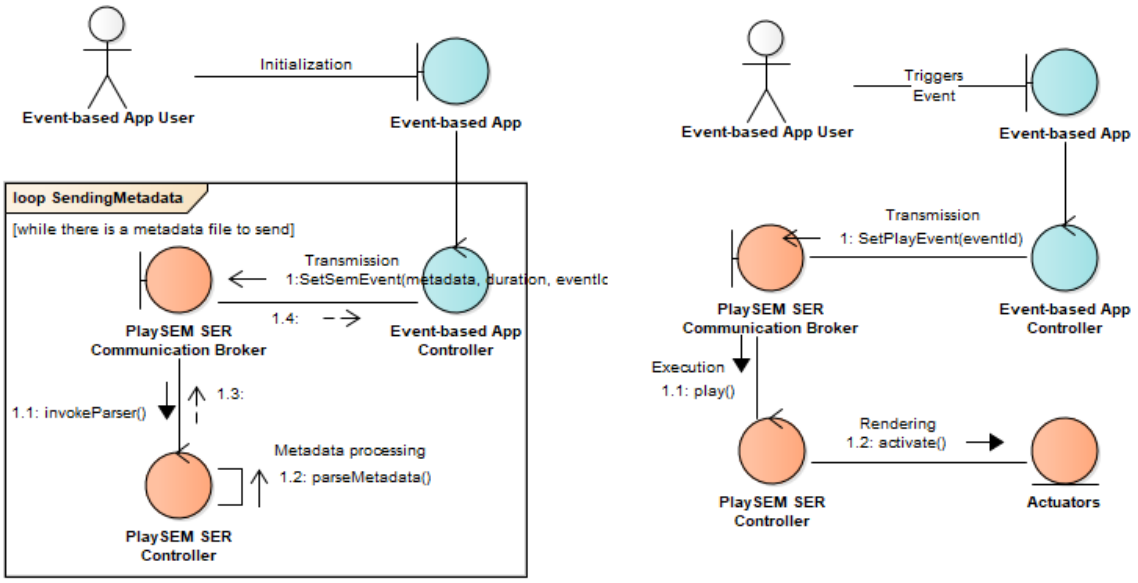

Figure 8: Interaction between event-based multimedia applications and PlaySEM SER 2 split in (a) loading process, and (b) interacting process.

4.8 Configuration Strategies

This section describes how to customize PlaySEM SER 2 to operate under different settings by simply setting it up accordingly. It also allows the system to have different behaviors dynamically without changing its code. All the settings are specified within the file *SERenderer.xml* which is explained in the following sections.

4.8.1 Initialization and Setup

Before starting PlaySEM SER 2, the setup should be informed. The excerpt below shows which parameters should be adjusted.

```
<!-- Main configuration -->
   <communicationServiceBroker>upnpService</communicationServiceBroker><!-- COMMUNICATION INTERFACE TO
BE USED -->
   <metadataParser>mpegvParser</metadataParser><!-- STANDARD TO BE CONSIDERED WHEN PARSING METADATA -->
   <lightDevice>mockLight</lightDevice><!-- DEVICE FOR RENDERING LIGHT -->
   <windDevice>mockWind</windDevice><!-- DEVICE FOR RENDERING WIND -->
   <vibrationDevice>mockVibration</vibrationDevice><!-- DEVICE FOR RENDERING VIBRATION -->
   <scentDevice>mockScent</scentDevice><!-- DEVICE FOR RENDERING SCENT -->
   <debugMode>true</debugMode><!-- PRINT MESSAGES ON CONSOLE --> ...
```

The attribute *communicationServiceBroker* gives the choice between different communication protocols (see Section 4.4). An id such as *upnpService* should be filled in for this attribute. It also requires a strategy such as *mpegvParser* to parse metadata according to a sensory effects metadata standard. As PlaySEM SER 2 currently supports four types of sensory effects, the attributes *lightDevice*, *windDevice*, *vibrationDevice*, and *scentDevice* must be informed. Finally, the attribute *debugMode* specifies whether debug messages will be printed in the console whereby the system runs. The ids filled out in each attribute must be declared in the same file which contains the specification for the replaceable components used in the system (see Section 4.8.3).

4.8.2 Communication, Connectivity, and Temporal Aspects

By introducing a network and considering sensory effects metadata processing time in mulsemedia systems, a set of other hurdles emerge, such as network delay, jitter, packet loss, sensory effects metadata transmuting, timetable building, among others, which can have an impact on user QoE. Therefore, timing issues within the integration of different applications to deliver sensory effects must be considered so that they do not affect performance as portrayed in Section 3.2. Given the plurality of the users' acceptable times (Section 2.4), a mulsemedia system should be able to consider the delay introduced by different technologies. Furthermore, some sensory effects such as scent, wind, and flavor have a tendency to linger (in the atmosphere or on the tongue). Therefore, if the mulsemedia system is able to calibrate and adapt the delivery of the sensory effects accordingly, it will produce better results.

PlaySEM SER 2 takes it into consideration by allowing a manual adjustment in time considering the technologies used to deliver the sensory effects. This is processed in execution time when the processing is being performed. The system will work on its main timetable to compensate for potential delays introduced by the components chosen for a specific profile. Within the declaration of each device, communication, and connectivity protocol, there is an attribute called *Delay* as shown in the following excerpt. It shall be adjusted in milliseconds. To this end, times provided in the literature and in devices specifications should be considered.

```
    <device>
       <id>exhaliaScentDevice</id>
       ...
       <properties>
          <delay>800</delay><!-- DEFINE IT IN MS -->
       </properties>
    </device>
```

4.8.3 Protocols and Properties

The declaration of the supported communication protocols, devices, and connectivity protocols and the addition of new ones as a way of expanding the support for new technology in PlaySEM SER 2 are linked to the file *SERenderer.xml*. The following excerpt shows how to describe a device and a connectivity type and how to connect one to another. The attribute *devices* represents a list of supported devices. A *device* must be described by filling in its *id*, used in the main configuration (see Section 4.8.1), *deviceClass*, for dynamic instantiation, and *connectivityInterface*, which refers to an id of a predeclared connectivity interface. The list of *connectivityInterfaces* is composed of the supported connectivity protocols, represented by an *id*, to identify the protocol, a *connectivityInterfaceClass*, for dynamic instantiation, if it requires *earlyConnection*, which indicates if the connection will be established when opening the system or on demand, and finally, a list of *properties*, which may contain zero or more properties to be used by the specific class that implements a connectivity protocol (see Section 4.6). For instance, one could indicate which slot within a scent machine contains a determined aroma.

```
<!-- Supported devices -->
   <devices>
   <!-- Arduino Devices (Light, wind, vibration) -->
      <device>
         <id>arduinoLightDevice</id>
         <deviceClass>
         br.ufes.inf.lprm.sensoryeffect.renderer.device.arduino.ArduinoLightDevice
         </deviceClass>
         <connectivityInterface>Serial</connectivityInterface><!—SET CONNECTIVITY INTERFACE -->
      </device> ...
<!-- Connectivity interfaces -->
   <connectivityInterfaces>
      <connectivityInterface>
         <id>Serial</id><!-- CONNECTIVITY MUST BE DECLARED TO BE USED -->
         <connectivityInterfaceClass>
         br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.serial.SerialOutput
         </connectivityInterfaceClass>
         <earlyConnection>true</earlyConnection>
         <properties>
            <serialPort>COM3</serialPort>
            <baudRate>9600</baudRate>
         </properties>
      </connectivityInterface> ...
```

The same pattern applies to the communication services and metadata parsers. There is a list of *communicationServices* that registers all the currently supported communication protocols. An *id* should be provided to identify the scheme, and a *communicationServiceClass*, for dynamic instantiation. The same goes for the catalog of *metadataParsers*. Section 4.8.1 shows how to set a strategy for each one before starting the system.

```
<!-- Communication services -->
   <communicationServices>
      <communicationService>
         <id>coapService</id>
         <communicationServiceClass>
         br.ufes.inf.lprm.sensoryeffect.renderer.service.coap.SERendererCoapService
         </communicationServiceClass>
      </communicationService> ...
   </communicationServices>
<!-- Metadata parsers -->
   <metadataParsers>
      <metadataParser>
         <id>mpegvParser</id>
         <metadataParserClass>
         br.ufes.inf.lprm.sensoryeffect.renderer.metadata.parser.mpegv.MPEGVSEMParser
         </metadataParserClass>
      </metadataParser>
   </metadataParsers> ...
```

4.9 Debug and Simulation

In order to check the integration between multimedia applications and PlaySEM SER 2, a mechanism for aiding the developers to find defects is offered. Debug refers to the process of finding issues with the code and repairing them. In the context of PlaySEM SER 2, this mechanism outputs messages from key moments such as receiving data and processing to follow up with the execution. It makes the process of discovering problems easier, finding which part of the code it refers to, and fixing the problems. It can be switch on/off by configuration before the system runs (see Section 4.8.1).

Another mechanism to find problems and correcting them is through the use of mock classes. They work like virtual devices and use the console as a mean of connectivity to print the outcome of the commands. For instance, instead of releasing a real scent at a specific time of a movie, a scent mock class would print the aroma and its intensity at this point. It is also useful to simulate the results of a metadata file. Nonetheless, new mock classes could be created to send commands to virtual environments that simulate the sensory effects. PlaySEM SER 2 provides by default one mock class for each one of the supported types of effects (see Section 4.5).

4.10 How is PlaySEM SER 2 Linked to the Requirements?

All the elements described through this section aim at introducing a robust set of elements to deal with the requirements described in Section 3. First, PlaySEM SER 2 follows a layered architecture to fit in with new technologies without making alterations to adjacent layers. This satisfies requirement *C3R1—Changeable*.

Design patterns are applied to its code mainly to provide reuse of components, to define uniform access with a high-level of abstraction for multimedia applications, and to allow different strategies within a certain profile without creating internal conditionals. As a result, the patterns described in Section 4.2 help to meet the requirements *C1R1—Multifunctional*, *C1R2—Compatible*, *C3R1—Changeable*, *C3R2—Extensible*, *C3R3—Orthogonal*, and *C3R4—Adaptable*. The programming language platform used to code PlaySEM SER 2, Java, has to do with requirement *C1R3—Portable*. Rationales are presented in Section 4.3.

Furthermore, the components of PlaySEM SER 2 play an important role in meeting the emerging requirements. For instance, the communication broker detailed in Section 4.4, caters for requirements *C1R1—Multifunctional*, as it provides singular operations for different types of multimedia applications; *C1R2—Compatible*, since it has reusable implementations of services for different communication protocols; *C2R1—Reliable*, once PlaySEM SER 2 relies on protocols that guarantee delivery; and *C2R2—Responsive*, considering that some of its implemented protocols are designed to be lightweight and therefore quick to react, as required by event-based multimedia applications.

Requirements met in the sensory effects processing layer (Section 4.5) are related to *C1R4—*

Compliant/Interoperable, to exchange data with other applications and components using standardized protocols; *C3R1—Changeable*, *C3R2—Extensible* and *C3R3—Orthogonal*, as it allows to extend the framework to support other devices, protocols, and standards, not meddling in neighbor layers combining different elements.

The connectivity interface described in Section 6 goes towards requirements *C1R2—Compatible*, *C1R4—Compliant/Interoperable*, *C2R1—Reliable*, *C3R1—Changeable*. The rationale is somewhat similar to those of the communication broker except that it deals with devices, not user applications. Finally, in Section 4.8, configuration strategies are taken into account to meet requirement *C3R4—Adaptable*, whereby adaptations for different scenarios of usage are made possible.

5 Case Studies on Variability with PlaySEM SER 2

Saleme et al. [49] presented different scenarios of usage in which evolutions of PlaySEM SER have dealt with variability in mulsemedia systems. With an emphasis on PlaySEM SER 2, we introduce two case studies where the current framework has been used to cope with heterogeneity at different levels. Furthermore, details on their configurations are also described. The aim is to make evident PlaySEM SER 2's capability of adapting to different conditions changing its parameters.

5.1 Video Clip Enriched with External Light, Smell, Vibration, and Wind

This mulsemedia system is composed of DIY (Do It Yourself) devices, smartphones, and a commercial scent device called Vortex Activ. A video clip annotated with MPEG-V is played on the PlaySEM SE Video Player and displayed on a monitor. Its sensory effects are provided to PlaySEM SER 2 following the interaction described in Section 4.7.1. Figure 9 shows the output devices in the environment. On the left side, there are seven elements that work together using heterogeneous technologies to deliver audiovisual (monitor and headphones), lighting (LED strip), wind (fan 1 and 2), vibration (two smartphones), and scent (Vortex Activ) effects. On the right side, the sensory effects are the same. However, the output comes from different devices using other technologies, that is, lighting is displayed on two smartphones whereas vibration is delivered by just one smartphone.

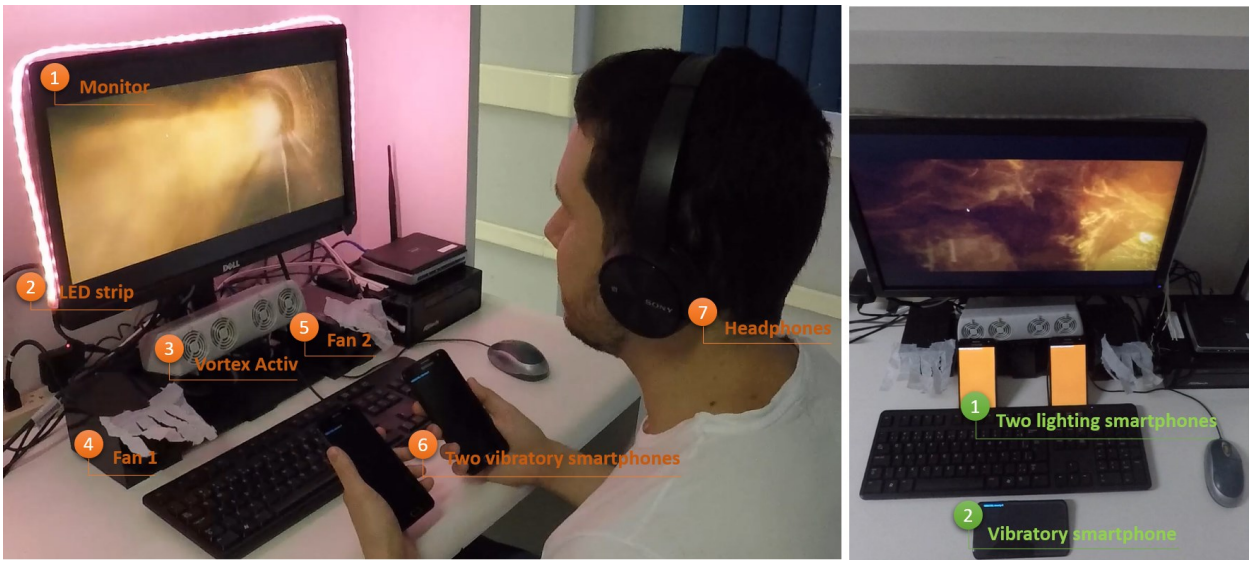

Figure 9: Video clip enhanced with lighting, wind, vibration, and scent effects. On the left side, the first set of devices to deliver these sensory effects. On the right side, similar devices that deliver the same effects using different technologies.

The connection with audiovisual devices is managed by the multimedia application whereas the sensory effects are of PlaySEM SER 2's responsibility. The following excerpt of PlaySEM SER 2's configuration file presents a list of parameters that will be instantiated for the scenario of usage on the left side of Figure 9. This shall be set up

as described in Section 4.8.1. The communication between the multimedia application and PlaySEM SER 2 is established via UPnP and the metadata parser used is to deal with MPEG-V.

```
<!-- Main configuration -->
   <communicationServiceBroker>upnpService</communicationServiceBroker>
   <metadataParser>mpegvParser</metadataParser>
   <lightDevice>arduinoLightDevice</lightDevice>
   <windDevice>arduinoWindDevice</windDevice>
   <vibrationDevice>sedroidVibrationDevice</vibrationDevice>
   <scentDevice>vortexScentDevice</scentDevice>
```

The devices provided in the initial configuration come from the next excerpt. The Arduino-based light and wind device use a serial connectivity interface with PlaySEM SER 2. The two smartphones that deliver vibrations are connected via Bluetooth whereas the scent device uses a proprietary serial connectivity interface (SerialFTD2XX) for chips manufactured by FTDI, which is embedded in Vortex Activ.

```
<!-- Lighting device -->
   <device>
      <id>arduinoLightDevice</id>
      <deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.arduino.ArduinoLightDevice
         </deviceClass>
      <connectivityInterface>Serial</connectivityInterface> ...
<!-- Wind device -->
   <device>
      <id>arduinoWindDevice</id>
      <deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.arduino.ArduinoWindDevice
         </deviceClass>
      <connectivityInterface>Serial</connectivityInterface> ...
<!-- Vibration device -->
   <device>
      <id>sedroidVibrationDevice</id>
      <deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.sedroid.SedroidVibrationDevice
         </deviceClass>
      <connectivityInterface>Bluetooth</connectivityInterface> ...
<!-- Scent device -->
   <device>
      <id>vortexScentDevice</id>

<deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.vortex.VortexScentDevice</deviceClass>
      <connectivityInterface>SerialFTD2XX</connectivityInterface> ...
```

For each connectivity interface, specific properties are indicated in order to supply PlaySEM SER 2 with information about where to send messages and which classes it should use to handle them. For instance, the serial connectivity uses port 'COM3' under baud rate '9600.' For the proprietary serial connectivity, an id (67330049) is provided to match to the scent device that is plugged. For the Bluetooth connection, the number of devices that will be connected and their URL (Uniform Resource Locator) for connection are provided. The smartphones are Android-based ones which run software that listens to connections and render sensory effects according to the messages received from PlaySEM SER 2.

```
<!-- Serial connectivity -->
   <connectivityInterface>
      <id>Serial</id>
      <connectivityInterfaceClass>br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.serial.
         SerialOutput</connectivityInterfaceClass>
      <properties>
         <serialPort>COM3</serialPort>
         <baudRate>9600</baudRate> ...
<!-- Serial FTD2XX -->
   <connectivityInterface>
      <id>SerialFTD2XX</id>
      <connectivityInterfaceClass>br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.serial.
         FTD2XXOutput</connectivityInterfaceClass>
      <properties>
         <device01-id>67330049</device01-id> ...
```

```
<!-- Bluetooth -->
   <connectivityInterface>
      <id>Bluetooth</id>
      <connectivityInterfaceClass>br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.bluetooth.
         BluetoothOutput</connectivityInterfaceClass>
      <properties>
         <number-of-devices>2</number-of-devices>
         <device01-connection-url>btspp://A816D0063584:5;authenticate=false;encrypt=false;master=false
            </device01-connection-url>
         <device02-connection-url>btspp://A816D006351E:5;authenticate=false;encrypt=false;master=false
            </device02-connection-url> ...
```

The following fragment of PlaySEM SER 2's configuration file shows the changes in the scenario of usage on the right side of Figure 9. Just as in the previous scenario, both the light and vibration devices are Android-based smartphones.

```
<!-- Main configuration -->
   <lightDevice>sedroidLightDevice</lightDevice>
   <vibrationDevice>sedroidVibrationDevice</vibrationDevice>
```

Whereas in the previous scenario just Bluetooth was used to connect to the smartphones, under this scenario Wi-Fi is configured to deliver lighting effects as follows.

```
<!-- Lighting device -->
   <device>
      <id>sedroidLightDevice</id>
      <deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.sedroid.SedroidLightDevice
         </deviceClass>
      <connectivityInterface>WiFi</connectivityInterface> ...
<!-- Vibration device -->
   <device>
      <id>sedroidVibrationDevice</id>
      <deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.sedroid.SedroidVibrationDevice
         </deviceClass>
      <connectivityInterface>Bluetooth</connectivityInterface> ...
```

Analogous to the first scenario, for each connectivity interface, communication parameters are supplied. The following excerpt presents parameters for the WiFi connection that requires the number of devices and the address of them. For Bluetooth, as a different device was used to the former case, a new URL was provided and just one device was set up.

```
<!-- Wi-Fi -->
   <connectivityInterface>
      <id>WiFi</id>
      <connectivityInterfaceClass>br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.wifi.
         WiFiOutput</connectivityInterfaceClass>
      <properties>
         <number-of-devices>2</number-of-devices>
         <device01-address>90.0.0.102:8080</device01-address>
         <device02-address>90.0.0.107:8080</device02-address> ...
<!-- Bluetooth -->
   <connectivityInterface>
      <id>Bluetooth</id>
      <connectivityInterfaceClass>br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.bluetooth.
         BluetoothOutput</connectivityInterfaceClass>
      <properties>
         <number-of-devices>1</number-of-devices>
         <device01-connection-url>btspp://4088058F7E2E:5;authenticate=false;encrypt=false;master=false
            </device01-connection-url> ...
```

This case study comes to light to show the flexibility of PlaySEM SER 2 in making slight changes in its configuration to accommodate different profiles of usage with heterogeneous interfaces for connectivity. This has proven useful to take advantage of the capabilities of devices such as smartphones to provide sensory effects. Furthermore, this made evident the needlessness to either change the framework's code or install additional SDKs

and APIs that eventually could be required for some devices.

5.2 Smell-intensive System

This scenario of usage involves just one scent device to deliver different intensities of smell—Exhalia SBi4. This is another proprietary device that is managed by PlaySEM SER 2. However, the framework goes beyond the SDK provided by Exhalia and provides a mechanism to switch on all of its four fans that blow scent crystals to deliver the smell. Figure 10 shows this environment. There is a monitor that presents a video clip annotated with MPEG-V running on PlaySEM SE Video Player and a scent device. PlaySEM SER 2 listens to the video player, processes SEM, and deals with the scent device in this case. As in the former section, the interaction between the multimedia application and PlaySEM SER 2 is as described in Section 4.7.1.

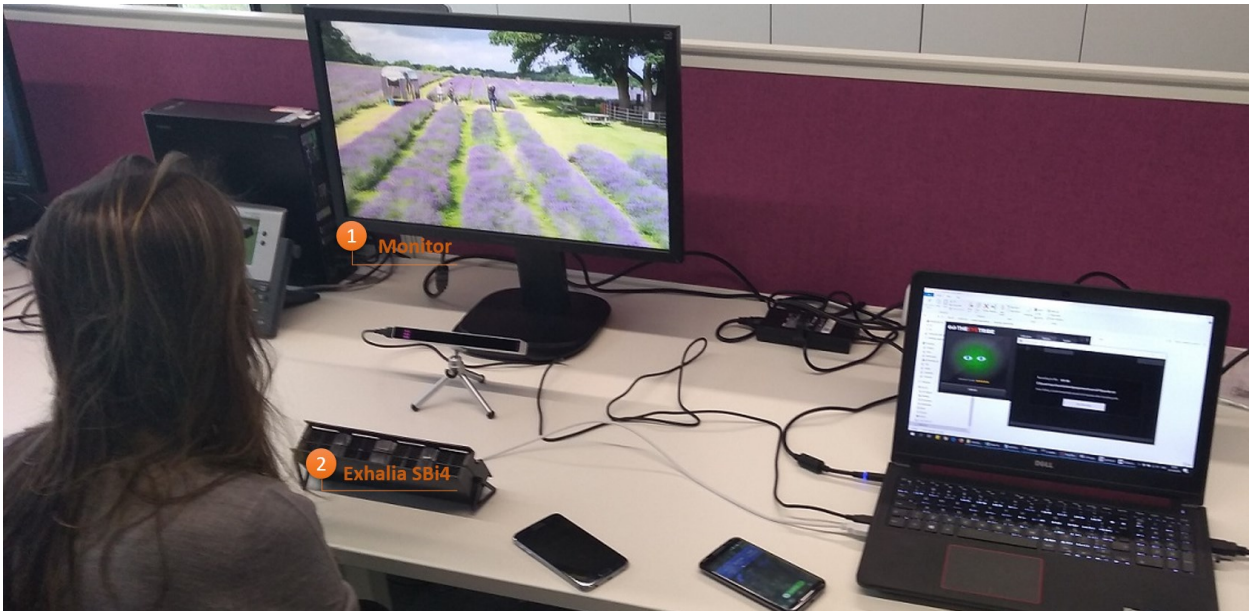

Figure 10: Smell-intensive system—audiovisual content along with scent delivered through four emitters at the same time using USB connectivity.

The following fragment displays Exhalia SBi4 set up for the initialization in conformance with what is described in Section 4.8.1. The other devices apart from the scent device are mock classes that exhibit the outcome in an output console if their sensory effect types are included in the presentation.

```
<!-- Main configuration -->
   <communicationServiceBroker>upnpService</communicationServiceBroker>
   <metadataParser>mpegvParser</metadataParser>
   <lightDevice>mockLight</lightDevice>
   <windDevice>mockWind</windDevice>
   <vibrationDevice>mockVibration</vibrationDevice>
   <scentDevice>exhaliaScentDevice</scentDevice>
```

Exhalia SBi4 is connected to the framework through a USB interface as parameterized in the next excerpt. A property called 'increasedIntensity' determines whether the scent device will emit a specific scent considering all the slots for cartridges for increased intensity. In case it is 'false,' the scent device works with one cartridge of scent at a time under the same intensity.

```
<!-- Scent device -->
   <device>
      <id>exhaliaScentDevice</id>
      <deviceClass>br.ufes.inf.lprm.sensoryeffect.renderer.device.exhalia.ExhaliaScentDevice
         </deviceClass>
      <connectivityInterface>Usb</connectivityInterface>
```

```
        <properties>
           <increasedIntensity>true</increasedIntensity> ...
```

Then, USB properties to reach and control Exhalia SBi4 are given. They should be collected from the operating system and then set up properly on PlaySEM SER 2 as follows.

```
<!-- USB -->
   <connectivityInterface>
      <id>Usb</id>
      <connectivityInterfaceClass>br.ufes.inf.lprm.sensoryeffect.renderer.connectivity.usb.UsbOutput
         </connectivityInterfaceClass>
      <properties>
         <vendorId>0x1781</vendorId>
         <productId>0x07D0</productId>
         <usbInterface>0</usbInterface>
         <inEndPoint>81</inEndPoint>
         <outEndPoint>1</outEndPoint>
         <timeout>5000</timeout> ...
```

Just like in the previous section, the framework rules out the use of SDKs and APIs to control the proprietary device in the smell-intensive system. By adding a mechanism not supported by the supplier to emit scent through all of the cartridges at the same time, it increases the utility of the device. Furthermore, from the preceding case study to this one, the only difference with regard to PlaySEM SER 2 is that a configuration profile was created to meet the specific needs for this scenario of usage, discarding code changes.

6 Conclusion and Future Directions

Although mulsemedia systems can be built just for specific purposes, they have an inherently heterogeneous trait, which makes it difficult to cope with many different aspects such as input integration from different multimodal interfaces, delay, responsiveness, sensory effects intensities, wearable and other varied devices for delivering sensory effects. To the extent that technology for delivering sensory effects evolves, changeless systems would become disposable, hence costly. Therefore, in this article, we identified hurdles that have left gaps and shortcomings in the realm of mulsemedia delivery. Then, we delineated the challenges and requirements to overcome these hurdles.

The solution presented to meet the requirements linked to the challenges is a framework—PlaySEM SER 2. It aims at presenting a flexible solution for delivering sensory effects to heterogeneous systems. This framework supports multi-communication and multi-connectivity protocols, multi-standards, and allows the accommodation of new technology relying on its set of architectural and design patterns to be applied successfully. It provides services to mulsemedia applications through a transparent communication broker and is able to work with timeline and event-based applications. Moreover, it presents a set of configurable parameters to customize the solution according to the needs imposed by mulsemedia applications and offers a mechanism to calibrate delay for each component it works with. In order to test new implementations on top of PlaySEM SER 2, it also provides a debug and flexible simulation scheme.

The works of Santos et al. [53], Saleme et al. [46], Saleme et al. [50], and Jalal et al. [25] have presented results on the use and evolution of PlaySEM SER until the current version, making evident its flexibility, responsiveness, and adaptability to work with different variables. Nevertheless, we provided two case studies whereby PlaySEM SER 2 is configured to be used in different scenarios of usage, discarding code changes. They demonstrated how it is possible to accommodate different profiles of usage featuring heterogeneous devices with different connectivities taking advantage of the framework's capabilities.

PlaySEM SER 2 does not constrain the design of mulsemedia systems. Although the framework deals with systems heterogeneity internally, it is not antagonistic to SDKs and APIs. Therefore, PlaySEM SER 2 can accommodate them to control devices if necessary. However, it has some limitations. Currently, it does not support content/context adaptation according to user preferences. Alternatively, it could be solved inside mulsemedia applications themselves. Another way to cope with it would be through the implementation of standardization such as MPEG-V Part 2, which provides means to describe user preferences. However, an input mechanism for supporting user preferences would have to be provided by multimedia applications, whereas mulsemedia renderers would need to interpret this metadata and adapt rendering devices accordingly. Further issues are related to safety— it has not been tackled yet. Due to the fact that incorporation of taste in mulsemedia applications is still very much in its infancy, support for this kind of sensory effect has not been implemented yet. Nonetheless, as soon as work on this effect

advances, it can be added to PlaySEM SER 2 easily by following the design strategies described in this article. A more important issue is related to the state of awareness in mulsemedia systems. For the sake of example, the perception of heat in cold environments would be different from warmer environments. This implies the calibration of the environment by capturing its state through sensors and adjusting the sensory effects in execution time. All are worthy future pursuits.

References

[1] Adelstein, B.D., Lee, T.G., Ellis, S.R.: Head tracking latency in virtual environments: psychophysics and a model. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 47, pp. 2083–2087. SAGE Publications Sage CA: Los Angeles, CA (2003)

[2] Balzarotti, N., Baud-Bovy, G.: Hpge: An haptic plugin for game engines. In: International Conference on Games and Learning Alliance, pp. 330–339. Springer (2018)

[3] Banks, A., Gupta, R.: MQTT Version 3.1.1, OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (2014). Online; accessed 19 October 2017

[4] Bartocci, S., Betti, S., Marcone, G., Tabacchiera, M., Zanuccoli, F., Chiari, A.: A novel multimedia-multisensorial 4d platform. In: 2015 AEIT International Annual Conference (AEIT), pp. 1–6 (2015). DOI 10.1109/AEIT.2015.7415215

[5] Broy, M.: The 'grand challenge' in informatics: Engineering software-intensive systems. Computer **39**(10), 72–80 (2006). DOI 10.1109/MC.2006.358

[6] Buschmann, F., Henney, K., Schmidt, D.: Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing. John Wiley & Sons (2007)

[7] Canna, A., Prinster, A., Fratello, M., Puglia, L., Magliulo, M., Cantone, E., Pirozzi, M.A., Salle, F.D., Esposito, F.: A low-cost open-architecture taste delivery system for gustatory fmri and bci experiments. Journal of Neuroscience Methods **311**, 1 – 12 (2019). DOI https://doi.org/10.1016/j.jneumeth.2018.10.003

[8] Choi, B., Lee, E.S., Yoon, K.: Streaming media with sensory effect. In: Information Science and Applications (ICISA), 2011 International Conference on, pp. 1–6 (2011). DOI 10.1109/ICISA.2011.5772390

[9] Cho, H.Y.: Event-Based control of 4D effects using MPEG RoSE. Master's thesis, School of Mechanical, Aerospace and Systems Engineering. Korea Advanced Institute of Science and Technology. Master's Thesis (2010)

[10] Comsa, I., Trestian, R., Ghinea, G.: 360º mulsemedia experience over next generation wireless networks - a reinforcement learning approach. In: 2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX), pp. 1–6 (2018). DOI 10.1109/QoMEX.2018.8463409

[11] Conti, F.: The chai libraries. Tech. rep. (2003)

[12] Covaci, A., Zou, L., Tal, I., Muntean, G.M., Ghinea, G.: Is multimedia multisensorial? - a review of mulsemedia systems. ACM Computing Surveys **51**(5), 91:1–91:35 (2018). DOI 10.1145/3233774

[13] Dobbelstein, D., Rukzio, E., Herrdum, S.: Demonstration of inscent: A wearable olfactory display as an amplification for mobile notifications. In: Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers, UbiComp '17, pp. 229–232. ACM, New York, NY, USA (2017). DOI 10.1145/3123024.3123185

[14] European Committee for Interoperable Systems: Interoperability (2018). URL http://www.ecis.eu/ecis-interoperability/

[15] Fette, I., Melnikov, A.: The WebSocket Protocol. RFC 6455. https://tools.ietf.org/html/rfc6455 (2011). Online; accessed 19 October 2017

[16] Gallacher, C., Mohtat, A., Ding, S., Kövecses, J.: Toward open-source portable haptic displays with visual-force-tactile feedback colocation. In: Haptics Symposium (HAPTICS), 2016 IEEE, pp. 65–71. IEEE (2016)

[17] Galster, M., Avgeriou, P.: Chapter 6 - supporting variability through agility to achieve adaptable architectures.

In: M.A. Babar, A.W. Brown, I. Mistrik (eds.) Agile Software Architecture, pp. 139 – 159. Morgan Kaufmann, Boston (2014). DOI 10.1016/B978-0-12-407772-0.00005-8

[18] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)

[19] Ghinea, G., Timmerer, C., Lin, W., Gulliver, S.R.: Mulsemedia: State of the art, perspectives, and challenges. ACM Trans. Multimedia Comput. Commun. Appl. **11**(1s), 17:1–17:23 (2014). DOI 10.1145/2617994

[20] GitHut: GitHut - A small place to discover languages in GitHub (2019). URL https://githut.info/

[21] Groher, I., Weinreich, R.: Supporting variability management in architecture design and implementation. In: 2013 46th Hawaii International Conference on System Sciences, pp. 4995–5004 (2013). DOI 10.1109/HICSS.2013.505

[22] Howell, M.J., Herrera, N.S., Moore, A.G., McMahan, R.P.: A reproducible olfactory display for exploring olfaction in immersive media experiences. Multimedia Tools and Applications **75**(20), 12311–12330 (2016). DOI 10.1007/s11042-015-2971-0

[23] ISO/IEC/IEEE Systems and software engineering: ISO/IEC/IEEE Systems and software engineering – Architecture description. ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000) pp. 1–46 (2011). DOI 10.1109/IEEESTD.2011.6129467

[24] Itkowitz, B., Handley, J., Zhu, W.: The openhaptics/spl trade/ toolkit: a library for adding 3d touch/spl trade/ navigation and haptics to graphics applications. In: First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics Conference, pp. 590–591 (2005). DOI 10.1109/WHC.2005.133

[25] Jalal, L., Anedda, M., Popescu, V., Murroni, M.: Qoe assessment for broadcasting multi sensorial media in smart home scenario. In: 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), pp. 1–5 (2018). DOI 10.1109/BMSB.2018.8436875

[26] Kaklanis, N., Votis, K., Tzovaras, D.: Adding haptic feedback to web applications towards improving end-users' cognitive capabilities. In: 2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), pp. 245–249 (2015). DOI 10.1109/CogInfoCom.2015.7390599

[27] Karunanayaka, K., Johari, N., Hariri, S., Camelia, H., Bielawski, K.S., Cheok, A.D.: New thermal taste actuation technology for future multisensory virtual reality and internet. IEEE Transactions on Visualization and Computer Graphics **24**(4), 1496–1505 (2018). DOI 10.1109/TVCG.2018.2794073

[28] Kim, J.R., Osgouei, R.H., Choi, S.: Effects of visual and haptic latency on touchscreen interaction: A case study using painting task. In: World Haptics Conference (WHC), 2017 IEEE, pp. 159–164. IEEE (2017)

[29] Kim, S.K., Joo, Y.S.: Sensible media simulation in an automobile application and human responses to sensory effects. ETRI Journal **35**(6), 1001–1010 (2014). URL http://dx.doi.org/10.4218/etrij.13.2013.0038

[30] Kolsanov, A., Nazaryan, A., Ivaschenko, A., Kuzmin, A.: Intelligent sdk for 3d surgery simulation. In: 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), pp. 384–387 (2016). DOI 10.1109/CONFLUENCE.2016.7508148

[31] Kovatsch, M., Lanter, M., Shelby, Z.: Californium: Scalable cloud services for the internet of things with coap. In: 2014 International Conference on the Internet of Things (IOT), pp. 1–6 (2014). DOI 10.1109/IOT.2014.7030106

[32] Lin, Y., Yang, M., Lin, Y.: Low-cost 4d experience theater using home appliances. IEEE Transactions on Multimedia pp. 1–1 (2018). DOI 10.1109/TMM.2018.2876043

[33] Luque, F.P., Galloso, I., Feijoo, C., Mart1̆3053′fn, C.A., Cisneros, G.: Integration of multisensorial stimuli and multimodal interaction in a hybrid 3dtv system. ACM Trans. Multimedia Comput. Commun. Appl. **11**(1s), 16:1–16:22 (2014). DOI 10.1145/2617992

[34] Martin, S., Hillier, N.: Characterisation of the novint falcon haptic device for application as a robot manipulator. In: Australasian Conference on Robotics and Automation (ACRA), pp. 291–292. Citeseer (2009)

[35] McGookin, D., Escobar, D.: Hajukone: Developing an open source olfactory device. In: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, pp. 1721–1728. ACM (2016)

[36] Mistrik, I., Galster, M., Maxim, B.R.: Software Engineering for Variability Intensive Systems: Foundations and Applications. CRC Press (2019)

[37] Monks, J., Olaru, A., Tal, I., Muntean, G.M.: Quality of experience assessment of 3d video synchronised with multisensorial media components. In: 2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), pp. 1–6 (2017). DOI 10.1109/BMSB.2017.7986129

[38] Murray, N., Ademoye, O.A., Ghinea, G., Muntean, G.M.: A tutorial for olfaction-based multisensorial media application design and evaluation. ACM Comp. Surveys **50**(5), 67:1–67:30 (2017). DOI 10.1145/3108243

[39] Murray, N., Lee, B., Qiao, Y., Muntean, G.M.: The impact of scent type on olfaction-enhanced multimedia quality of experience. IEEE Transactions on Systems, Man, and Cybernetics: Systems **47**(9), 2503–2515 (2017). DOI 10.1109/TSMC.2016.2531654

[40] Nakamura, H., Miyashita, H.: Development and evaluation of interactive system for synchronizing electric taste and visual content. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12, pp. 517–520. ACM, New York, NY, USA (2012). DOI 10.1145/2207676.2207747

[41] Pimentel, V., Nickerson, B.G.: Communicating and displaying real-time data with websocket. IEEE Internet Computing **16**(4), 45–53 (2012). DOI 10.1109/MIC.2012.64

[42] Pressman, R., Maxim, B.: Software Engineering: A Practitioner's Approach, 8 edn. McGraw-Hill Science/Engineering/Math (2014)

[43] Rainer, B., Waltl, M., Cheng, E., Shujau, M., Timmerer, C., Davis, S., Burnett, I., Ritz, C., Hellwagner, H.: Investigating the impact of sensory effects on the quality of experience and emotional response in web videos. In: 2012 Fourth International Workshop on Quality of Multimedia Experience, pp. 278–283 (2012). DOI 10.1109/QoMEX.2012.6263842

[44] Ranasinghe, N., Do, E.Y.L.: Digital lollipop: Studying electrical stimulation on the human tongue to simulate taste sensations. ACM Transactions on Multimedia Computing, Comm., and Applications (TOMM) **13**(1), 5 (2017)

[45] Ranasinghe, N., Nguyen, T.N.T., Liangkun, Y., Lin, L.Y., Tolley, D., Do, E.Y.L.: Vocktail: A virtual cocktail for pairing digital taste, smell, and color sensations. In: Proc. of the 2017 ACM on Multimedia Conference, pp. 1139–1147. ACM (2017)

[46] Saleme, E.B., Celestrini, J.R., Santos, C.A.S.: Time Evaluation for the Integration of a Gestural Interactive Application with a Distributed Mulsemedia Platform. In: Proceedings of the 8th ACM on Multimedia Systems Conference - MMSys'17, pp. 308–314. ACM Press, New York, New York, USA (2017). DOI 10.1145/3083187.3084013

[47] Saleme, E.B., Covaci, A., Mesfin, G., Santos, C.A.S., Ghinea, G.: Mulsemedia DIY: A Survey of Devices and a Tutorial for Building your own Mulsemedia Environment. ACM Computing Surveys (2019). DOI 10.1145/3319853 . In press.

[48] Saleme, E.B., Santos, C.A.S., Falbo, R.A., Ghinea, G., Andres, F.: Towards a reference ontology on mulsemedia systems. In: Proceedings of the 10th International Conference on Management of Digital EcoSystems, MEDES '18, pp. 23–30. ACM, New York, NY, USA (2018). DOI 10.1145/3281375.3281378

[49] Saleme, E.B., Santos, C.A.S., Ghinea, G.: Coping with the challenges of delivering multiple sensorial media. IEEE MultiMedia p. 11 (2018). DOI 10.1109/MMUL.2018.2873565

[50] Saleme, E.B., Santos, C.A.S., Ghinea, G.: Improving response time interval in networked event-based mulsemedia systems. In: Proceedings of the 9th ACM Multimedia Systems Conference, MMSys '18, pp. 216–224. ACM, New York, NY, USA (2018). DOI 10.1145/3204949.3204965

[51] Saleme, E.B., Santos, C.A.S.: PlaySEM: a Platform for Rendering MulSeMedia Compatible with MPEG-V. In: Proceedings of the 21st Brazilian Symposium on Multimedia and the Web - WebMedia '15, pp. 145–148. ACM Press, New York, New York, USA (2015). DOI 10.1145/2820426.2820450

[52] Sanfilippo, F., Weustink, P.B.T., Pettersen, K.Y.: A coupling library for the force dimension haptic devices and the 20-sim modelling and simulation environment. In: IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, pp. 000168–000173 (2015). DOI 10.1109/IECON.2015.7392094

[53] Santos, C.A.S., Neto, A.N.R., Saleme, E.B.: An Event Driven Approach for Integrating Multi-sensory Effects to Interactive Environments. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics, pp. 981–986. IEEE (2015). DOI 10.1109/SMC.2015.178

[54] Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252. https://tools.ietf.org/html/rfc7252 (2014). Online; accessed 19 October 2017

[55] Sommerville, I.: Software Engineering, 10th edn. Pearson (2015)

[56] Suk, C.B., Hyun, J.S., Yong, L.H.: Sensory effect metadata for smmd media service. In: M. Perry, H. Sasaki, M. Ehmann, G.O. Bellot, O. Dini (eds.) ICIW, pp. 649–654. IEEE Computer Society (2009). DOI 10.1109/ICIW.2009.104

[57] Sulema, Y.: Asampl: Programming language for mulsemedia data processing based on algebraic system of aggregates. In: M.E. Auer, T. Tsiatsos (eds.) Interactive Mobile Communication Technologies and Learning, pp. 431–442. Springer International Publishing, Cham (2018)

[58] UPnP Forum: UPnP Device Architecture 1.0. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf (2008). Online; accessed 05 April 2018

[59] Vi, C.T., Marzo, A., Ablart, D., Memoli, G., Subramanian, S., Drinkwater, B., Obrist, M.: Tastyfloats: A contactless food delivery system. In: Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces, ISS '17, pp. 161–170. ACM, New York, NY, USA (2017). DOI 10.1145/3132272.3134123

[60] Waltl, M., Rainer, B., Timmerer, C., Hellwagner, H.: An end-to-end tool chain for sensory experience based on mpeg-v. Image Commun. **28**(2), 136–150 (2013). DOI 10.1016/j.image.2012.10.009

[61] Waltl, M., Timmerer, C., Hellwagner, H.: A test-bed for quality of multimedia experience evaluation of sensory effects. In: 2009 International Workshop on Quality of Multimedia Experience, pp. 145–150 (2009). DOI 10.1109/QOMEX.2009.5246962

[62] Waltl, M., Timmerer, C., Hellwagner, H.: Improving the quality of multimedia experience through sensory effects. In: 2010 Second International Workshop on Quality of Multimedia Experience (QoMEX), pp. 124–129 (2010). DOI 10.1109/QOMEX.2010.5517704

[63] Wegner, P.: Interoperability. ACM Comput. Surv. **28**(1), 285–287 (1996). DOI 10.1145/234313.234424

[64] Yoon, K., Kim, S.K., Han, J.J., Han, S., Preda, M.: MPEG-V: Bridging the Virtual and Real World, 1st edn. Academic Press (2015)

[65] Yoon, K.: End-to-end framework for 4-d broadcasting based on mpeg-v standard. Image Commun. **28**(2), 127–135 (2013). DOI 10.1016/j.image.2012.10.008

[66] Yuan, Z., Bi, T., Muntean, G.M., Ghinea, G.: Perceived synchronization of mulsemedia services. IEEE Transactions on Multimedia **17**(7), 957–966 (2015). DOI 10.1109/TMM.2015.2431915

[67] Yuan, Z., Chen, S., Ghinea, G., Muntean, G.M.: User quality of experience of mulsemedia applications. ACM Trans. Multimedia Comput. Commun. Appl. **11**(1s), 15:1–15:19 (2014). DOI 10.1145/2661329

[68] Yuan, Z., Ghinea, G., Muntean, G.M.: Beyond multimedia adaptation: Quality of experience-aware multi-sensorial media delivery. IEEE Transactions on Multimedia **17**(1), 104–117 (2015). DOI 10.1109/TMM.2014.2371240