Transferring an Interactive Display Service to the Virtual Reality

Jens Naber, Christian Krupitzer, Christian Becker

University of Mannheim
Schloss, 68161 Mannheim, Germany
{jens.naber, christian.krupitzer, christian.becker}@uni-mannheim.de

***Abstract*—In today's world, smart devices – such as laptops, smartphones, or smart watches – may substitute the need for physical presence of people. By offering a virtual representation, these devices support social interactions, e.g., telecommuting, remote studying through e-learning, or long-distance relationships. With audio and video streaming platforms like Skype, Apple FaceTime, or Google Hangouts it is possible to stay in contact with friends and colleagues. However, these platforms only allow communication between two peers or at most in very small groups. Additionally, they are very inflexible regarding content sharing and support only limited, defined use cases. In this paper, we present an extension to our pervasive display service for a remote virtual environment. This allows an immersive participation at remote activities in larger groups, while retaining the flexibility regarding joining devices of a pervasive middleware. Our contributions are threefold. First, we present a reusable design for integrating an interactive display service into virtual environments. Second, we implement our approach using the BASE middleware and the Unity 3D engine. Third, we evaluate our approach in a smart meeting room scenario.**

I. Introduction

Today, electronic devices are ubiquitous in our society. Most people own multiple smart devices to support their daily work and private life. With the development of pervasive middlewares these devices are getting highly integrated into our everyday tasks. Whereas one challenge is the integration of these devices into the workflow of businesses or the teaching at universities, another issue is the mobility of users. Especially in office jobs, many people work remotely from business trips or at home. Smart meeting and lecture rooms are developed, which allow all participants to incorporate their own devices and content as well as participate remotely in meetings, lectures, or conferences. However, the physical world – e.g., the lecture hall – and the virtual representation – e.g., a virtual lecture room – are often not well integrated due to synchronization issues as well as heterogeneous resources and content. To cope with this challenge, it might be beneficial to connect remote users and local users in the physical location to a pervasive middleware. Such a middleware offers various benefits as it handles service discovery and ad-hoc connections of devices, as well as abstraction from resource heterogeneity.

In this paper, we integrate these strengths in our approach and propose a system model on top of such a pervasive middleware. The approach extends our pervasive display service from [1] for virtual environments. In [1], the middleware allows applications from several devices to use pervasive displays and share content among users or just use the comfort of a larger screen size. Users and applications can access and manipulate content on the public screen based on user and sensor inputs. In this paper, we extend this concept to a virtual environment. Users and applications may seamlessly interact with services and users in the physical as well as virtual world. The flexibility of our pervasive middleware is also key to the virtual extension. Participants are able to connect with a high variety of devices, including their PC or Laptop, smartphones, as well as more specific devices, especially headmounted virtual reality displays like *Oculus Rift* or *HTC Vive*. In the virtual reality, we achieve a very immersive experience, which can come close to being at the location. With this service, it is possible to communicate and share content between remote participants. Due to the high flexibility, the middleware and services are capable to support a variety of scenarios. The most obvious use cases would be *smart lecture* or *meeting rooms*, where users share content and presentations in a mutual environment. However, other scenarios can benefit from remote participation, such as entertainment events (e.g., concerts or sports events) or conferences. Our contributions are threefold. First, we present a reusable design that extends our approach from [1]. Second, we implement the system using the BASE middleware [2] and the Unity game engine. Third, we evaluate our system in a smart meeting room scenario.

In the following, we first discuss related work in Section II, before we describe the system model in Section III. In Section IV, we define our requirements for the Virtual Display Service and present its design in Section V. Afterwards, we describe the implementation of our prototype in Section VI and the evaluation in Section VII. We close the paper with a conclusion and an outlook on future work.

II. Related Work

Different commercial products and research prototypes enable the enhancement of long distance communication and working. There are specialized, commercial solutions for text messaging (e.g., Whatsapp), video and audio streaming (e.g., Skype), or cooperative working and file sharing (e.g., Google Docs). While they are very popular for their specific application, they are not able to provide an integrated solution which offers various functionalities. Further, it might be possible that within a company or institute, different applications are used for the same purpose. Hence, users might need to use different applications for the same purpose if they are part of various teams or groups. This results in incompatibility between the applications. Additionally, only a suite of applications offers all needed functionality. Our approach should not offer all the functionality of the mentioned applications but focuses on the integration of different applications into a common platform.

By combining technologies, researchers and companies are trying to provide a more integrated workspace. The DOLPHIN [3] meeting support system started by using an interactive electronic whiteboard to enhance meeting rooms and making it accessible for remote computers. The ActiveTheatre System [4] provides a more data centered approach for supporting surgeons, nurses, and patients during the process of preparing and executing an operation. It is possible with this system to access and capture patients' data inside and outside of the operating room. A further example for a collaborative groupwork approach is GroupMind [5]. This mind-mapping system enables the shared work on one document, which is additionally shown on a big screen in the meeting room to display all the changes. Additionally, there are stand-alone collaborative groupwork solutions from different companies, like Cisco WebEx, IBM Connections, or Microsofts Sharepoint.

These systems represent the classical approach for collaborative working. In some cases, developers already tried to make the experience more immersive for the user by implementing their approach as a virtual environment. DIVE [6] provides a multi-user virtual environment, which focuses on the communication between humans. Based on the DIVE system the Virtual Society project of Lea *et al.* [7] developed a browser-based virtual environment for larger groups to chat and share information. It shows a conference room, in which the users can communicate via text or audio and use a virtual display for presentations. De Lucia *et al.* [8] use a different approach and extend the video game Second Life with a virtual campus. The campus provides a possibility for lecturers to give presentations and discuss with students in remote places. Guided by the lecturer, participants group themselves up and work together on shared learning tasks.

All of these approaches are restricted to specific use cases and often also to specific application environments or device platforms. As these systems are tailored for their specific use case, they require a high effort for reconfiguration to other use cases. Additionally, many of the approaches are limited in their functionality and do not combine all the possibilities of communication, content sharing, and interaction with the content. Therefore, in this paper, we propose a display service based on our pervasive middleware [1], as a combined solution for the physical and virtual space. It provides a flexible and immersive approach for sharing content, communication, and collaborating with people from remote places.

III. System Model

In [1], we present middleware-based display services for public and private pervasive displays. Our implementation supports user interaction as well as integrates sensor input at the remote device. We assume smart environments consisting of I) user devices, such as smartphones, and II) environment resources, especially visual output devices and different input capabilities. These devices are connected by a pervasive middleware. User applications can use devices in the user as well as the environment domain. According to the nature of smart, pervasive environments, we focus on adhoc smart peer groups. Figure 1 depicts this system approach. Within this paper, we extend the existing system model by a virtual world. Therefore, we map the structure presented in [1] – containing user domain, physical middleware, and environment domain, as well as the distinction of applications and services – for our pervasive display service to the virtual world. Additionally, a defined protocol enables a seamless connection between the pervasive middleware of the physical world and a corresponding intermediate in the virtual world. Hence, user and environment applications of the physical world can interact with user and environment devices in both worlds simultaneously. Users in the virtual environment are represented as moving and interacting avatars. Vice versa, virtual applications might use devices in the physical world. However, in this paper, we focus on interactions between applications in the physical world with user and environment devices in both worlds, physical and virtual one. This includes bi-directional communication between both worlds.

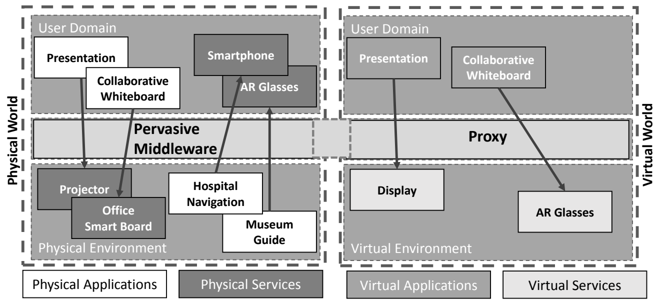

Fig. 1. Our System Model.

A smart meeting room is a good example where the benefits of such an approach supports users. Here, the user application is a presentation that uses services such as displays or projectors. However, as today's business world gets more and more global, it might be necessary to integrate business partners from other countries in the meetings. Therefore, a virtual presentation of the meeting room would enable remote attendance. Using our approach, the system would guarantee the synchronization of the presentation's content shown on devices in the physical meeting room as well as the virtual one. Further, our system would offer an approach for communication between both worlds. Following, we first present the requirements of the extension of the model to the virtual world. Thereafter, we present the design of our system.

IV. Requirements

Based on our system model and the analysis of the scenarios we described in [1] for the Virtual Display Service, we now derive our requirements for the implementation. As the system is based on our pervasive display service from [1], similar requirements exist. The system from [1] offers transfer of content to displays and the possibility to interact with them through user and sensor input. In this paper, we complement the interaction of users with virtual and physical services by functionality for communication between users. Hence, in the following, we present the requirements relevant to the extended system.

a) Content Type: To facilitate the use of the system in various application areas, a variety of content types has to be supported. For some applications like chat or e-mail, it is enough to transmit and display text. However, in more complex scenarios we need to enable the use of image or even video transmission. For instance, this allows presentations in a classroom environment or live transmissions of concerts. In addition to the *visual content*, we need the possibility for an *audio output*. The audio support allows to transmit the speech of a presenter together with the slides. In a concert scenario, the audio even provides the main content and is supported by the video transmission. Hence, the system should support the content types *text*, *image*, *video*, and *audio*.

b) User/Sensor Input: Users need to be able to directly interact with physical and virtual services. One example for such services are physical and virtual displays. While viewing a presentation on the virtual display, users need to be able to forward the slides. Similar, video transmissions may be paused or the user wants to rewind the video. To enable the interaction with displays in the virtual reality the service should enable mouse input at the display. Additionally, keyboard shortcuts can simplify the interaction with several virtual displays. The types of sensor and user input may vary for other services, e.g., a location-based service needs some sensor that recognizes the location (e.g., based on GPS).

c) Responsiveness: To ensure a smooth experience when connecting the real world with a virtual environment, the system has to minimize the latency between these two. If, for instance, the content of a presentation reaches users at different points in time, it hinders the communication between the users and the discussions of the topic.

d) Communication: The ability to share content in both worlds, the real but also in the virtual one, is not enough to replicate the experience. Users also have to be able to communicate with each other. As the middleware should allow collaborative work in an efficient manner, it has to integrate multiple different communication styles to eliminate the need of additional communication tools. The communication between the users is divided into three types: (i) textual communication, (ii) verbal communication, and (iii) gesture-based communication. First, the system should offer text for short, non-disruptive, and asynchronous communication. Second, it should support audio transmissions for enabling efficient group work through real-time communication. Last, the virtual environment should support gestures to communicate. Therefore, users can enter specific commands to enable gestures of their avatars in the virtual environment, e.g., raising the hand in a lecture.

e) Access Control: In some situations, the system may restrict the communication between users. Users in real world lectures or conference rooms mostly do not start to chat, while a presentation is running. In virtual environments, due to the partial anonymity and a higher amount of users, the communication is less controlled. To avoid this problem, an access control for the communication in the virtual environment should be integrated. This gives the presenter the possibility to enable and disable the communication in the virtual environment and between the virtual and real world. Additionally, it should be possible to allow the division of the users into different groups for efficient communication. The access control should support efficient grouping to divide potentially large amounts of users into teams.

V. Virtual Display Service

In this section, we present the design of our approach for integrating virtual and physical, pervasive services and devices. In the following, we focus on an interactive display service for virtual environments. This is integrated in our approach for pervasive displays from [1]. Further, it extends the approach by inter-user communication capabilities. Next, we present the overall architecture. After that, we show the designs of (i) the inter-user communication, (ii) access control, and (iii) the protocol for interactions between the physical and virtual world.

A. Architecture

The presented solution is based on our interactive display service [1] for applications in physical environments, which itself uses the pervasive BASE middleware [2] as a foundation. However, the encapsulated design enables the exchange of the middleware by other pervasive middlewares. Developers should be able to easily use the generic service for their applications. In this section, we design a more generic approach by not specifying the pervasive middleware that should be used. The pervasive middleware enables the connection between the user domain and the physical environment (see Figure 1). Further, it connects the physical and the virtual world. Our design approach uses an existing game engine for representing the virtual world. Following, we discuss the architecture of the extended display service and the connection between the pervasive middleware and the game engine. Figure 2 depicts the combination of the physical (PDS) and virtual display service (VDS). In this paper, we focus on the latter. For more information on the implementation of our PDS, the interested reader is referred to [1].

Our target application domain is the presentation of content in the virtual and the physical world, simultaneously, i.e., an application in the physical world should be able to present content on physical and virtual display. Further, interactions between users and the application (or its users) and inter-user communication should be possible. The pervasive middleware

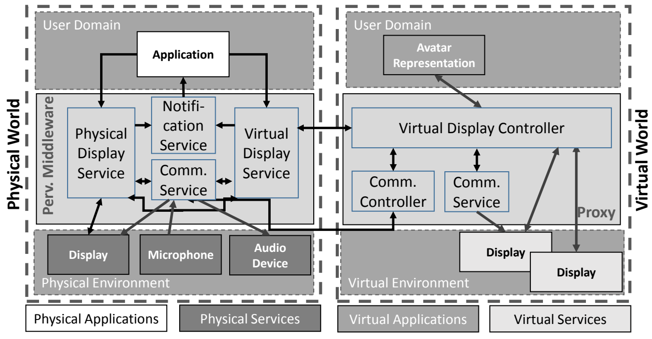

Fig. 2. Architecure of the interactive display service with the virtual environment extension.

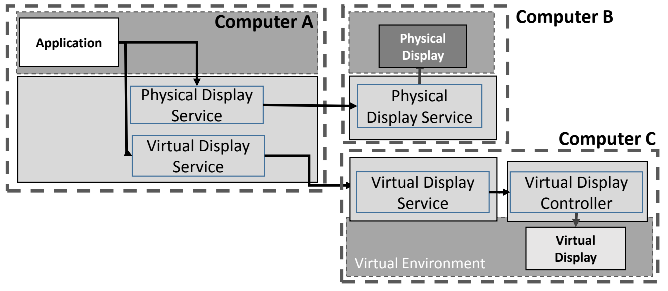

Fig. 3. Data flow of an application using a physical as well as a virtual display. Application, physical, and virtual display are running on separated computers. Notification Service and Communication Controller are omitted.

connects devices and services. The VDS, a part of the physical middleware, is connected with the Virtual Display Controller (VDC) which represents as part of the game engine the virtual world. Analogously, the PDS connects the application with physical devices. Both display services are part of the physical middleware. The services are responsible for communication with the user application as well as with other services. As we focus on the display services, we omit details of further elements of the physical middleware in the following. The application is able to mirror its content to physical and virtual displays, but it is also possible to provide all accessed displays with own customized content. For the communication with the VDC, the network functionality within the Game Engine is used. The VDC and the VDS use a defined protocol for their interaction, which is discussed in detail in Section V-D.

The VDC is responsible for administering the displays in the virtual environment and for routing user interaction and communication back to the application through the pervasive middleware. Multiple virtual displays can be connected and independently addressed by the VDC. The designer of the virtual environment can easily place new displays and the VDC identifies them by their IDs. Furthermore, the VDC propagates the amount of displays and their IDs to the application. Using the IDs, the application is able to send different content to the displays. Users in the virtual environment also have the possibility to interact directly with the displays. For instance, they can request the next slide of a presentation or pause a video. The virtual displays propagate these events to the VDC, which then informs the application via the VDS and the Notification Service. Figure 3 shows the data flow of one application that uses a physical display as well as a virtual one according to described the approach.

Additionally, the architecture contains a Communication Controller which is connected to the Communication Service in the physical middleware. Its main purpose is to orchestrate the communication between the users, therefore, it is responsible for the communication using input and output channels. We take a closer look on this part in the next section. To forward user interaction requests and communication information to the application, its Notification Service is used. Both, the PDS and VDS, are using the same Notification Service. The service is provided to the application developer as an interface. Each application has to handle the transmitted data itself and may ignore data that is currently not relevant. To get a more detailed explanation on the Notification Service please refer to [1].

B. Inter-User Communication

Additionally to the content output via the virtual displays, we also provide the possibility for users to communicate with other users within the virtual and physical environment via text, speech, and gestures. One part of the communication input is a UI-based classical text chat. The second part of the communication input uses the user's microphone for audio-based communication and keyboard shortcuts to trigger specific gestures. Furthermore, the virtual environment is able to output the audio stream of other users. The retrieved gestures are executed by the virtual avatar of the sending user. These functionalities are also designed independently from the specific virtual environment and can easily be transferred to new scenarios. The Communication Controller orchestrates the complete process and assigns incoming messages to the correct output device or respective user avatar. Additionally, the Communication Controller collects user input and prepares it for the transmission. For audio input, the raw data from the microphone has to be encoded and split up in chunks (decoded and assembled respectively for the incoming traffic) by the Communication Controller. In- and outgoing communication is routed to the Communication Service of the physical middleware, which is responsible for the connection with the communication devices, e.g., microphones, audio devices, and displays. Communication is not limited to the users in the virtual environment. Additionally, physical devices can show/ play the communication output for all users not connected to the virtual environment. Vice versa, users may use applications for their devices or physical displays for communicating with their peers in the virtual environment.

C. Access Control

The VDS offers a mechanism for access control. This allows to group users, which is helpful for creating teams to work on one task without disturbing other users or groups. The pervasive middleware uses the access control system to identify relevant users and forwards communication only to the target group. This reduces unnecessary traffic and only relevant messages have to be processed by the virtual environments. Furthermore, confidential content can be restricted to specific users. Each group is identified by a unique ID and the set of members. A user can be assigned to several groups and send messages to groups. Furthermore, the virtual environment saves a history for the text messages associated with each group, so that it is possible to read messages from the past after entering a new group. Each incoming audio stream is bound to the corresponding user avatar. With this method, it is possible to play audio messages from multiple users in parallel. Additionally, the audio streams are mixed based on the location of the avatar in the virtual environment.

In addition to the group mechanism, the VDS allows for a more fine-grained control of the communication between users. It is possible to limit the ability of a user to communicate with other users. The service can mute the user for the various communication types separately. For instance, this allows lecturers to mute students for avoiding chattering.

D. Protocol for Interaction

We defined a protocol for the interaction between the pervasive middleware and the VDC. The protocol header has the following structure:

```
struct MessageHeader {
  enum          requestType;
  enum          command;
  long          id;
  long          length;
};
```

The message's data is saved in the payload, which might be additionally structured depending on the message type. Figure 4 shows the message types with their corresponding direction of communication. We distinguish between device-related (`DEV`) and user-related messages (`USR`), indicated by the first header field. The subsequent header field describes the type of command. The following device-related commands are available: new content for the display (`OBJ`) and new user input or display control statement, respectively (`CTR`). If applications aim to interact with a virtual display, an ID that uniquely represents a display follows after the `OBJ`/ `CTR` command. Using this value, the VDC addresses the correct display. The other way around, the application can identify which display triggered an event. The following field of the message specifies the length of the payload. Afterwards, the payload follows. In case of a `OBJ` command, this is the content that should be shown on a display, e.g., an image, video, or text. Additionally, in case of a `CTR` command, either the application sends an instruction to the display or vice versa. On the one hand, the display service is able to transmit an event triggered by the user at the virtual display. This can be a click event or a pressed keyboard shortcut. On the other hand, the application can send a control command to the virtual display, e.g., a request to forward a presentation or pause a video.

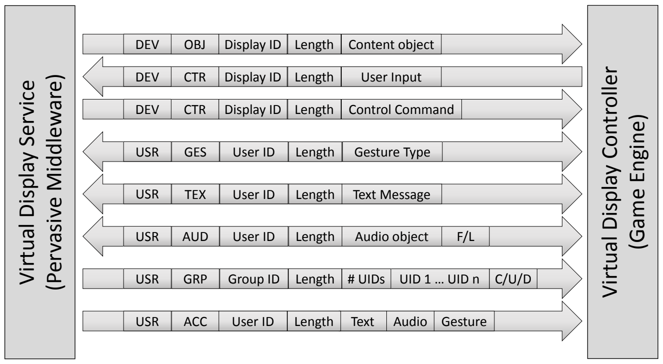
Virtual Display Service (Pervasive Middleware)
Virtual Display Controller (Game Engine)
DEV | OBJ | Display ID | Length | Content object
DEV | CTR | Display ID | Length | User Input
DEV | CTR | Display ID | Length | Control Command
USR | GES | User ID | Length | Gesture Type
USR | TEX | User ID | Length | Text Message
USR | AUD | User ID | Length | Audio object | F/L
USR | GRP | Group ID | Length | # UIDs | UID 1 … UID n | C/U/D
USR | ACC | User ID | Length | Text | Audio | Gesture

Fig. 4. Protocol for interaction between service and controller.

The user-related messages are designed to support bi-directional information flow between the physical applications and the virtual environment to enable inter-user communication. For the three communication styles, the protocol provides corresponding command types: `TEX` for text, `AUD` for audio, or `GES` for gestures. Each of them are followed by a unique user ID for identifying who send the message. Text and audio messages have a variable length and require therefore a transmission of the length before the payload. In the case of audio transmissions, we finish with a field to mark the first and last package for decoding purposes. For administration of groups, we added the `GRP` command. Each group needs a unique ID to identify it later for changes. Afterwards, the amount of users included in the group is transmitted, followed by the ID of each user. At the end, a flag is transmitted to indicate whether the group should be created, updated, or deleted. The access control command `ACC` disables the ability of users to communicate. It contains three boolean values to toggle the communication channels separately.

VI. Implementation

Based on the design in the previous section, this section describes the implementation of our approach. As physical middleware, we use the pervasive *BASE* middleware [2]. Its lightweight micro-broker architecture allows for an easy discovery and connection to the implemented services, while still keeping the overhead low. Additionally, it is designed to fit on a high variety of devices and supports a multitude of communication protocols. However, other pervasive middlewares may be used. Furthermore, we need to choose a specific 3D engine for implementation of the virtual environment. A game engine is a middleware that supports developers of (serious) games by offering reusable development artifacts, e.g., a 3D engine, world editor, or scripts for the game logic. Out of the in [9] presented game engines, we choose *Unity*[1] as it is flexible, offers an easy to learn editor, and supports different programming languages (JavaScript, Boo, and C#) as well as all major PC (including VR devices), mobile and console platforms. We implemented our virtual environment including the Communication Controller and the VDC in C#.

[1]Unity website: https://unity3d.com/

In the following, we focus on describing the implementation of (i) the connection between BASE and Unity, (ii) the implementation and usage of virtual displays in Unity, and (iii) the inter-user communication.

A. Extending the Physical Display Service with Virtuality

In this section, we focus on the implementation of the virtual displays as well as the connection between the virtual and the physical application, hence, the implementation of the VDS and the VDC. Unity offers so called `GameObjects`. These objects are containers that can contain various objects, such as scripts that implement an object's logic, audio content, or visual content. Hence, we implemented GameObjects for the different contents the virtual display should show. As Unity supports the formats JPEG, PNG, and OGG out of box, our display works with them. Additionally, using *Ghostscript*, the VDS converts PDF files to a collection of PNG files which can be shown on the virtual display. To reduce delays due to the conversion process of large PDF files, the conversion is done in a background process and converted pages are sent directly to the VDC. The VDS sends the relevant content to the VDC. Further, the VDC maps received content to virtual displays. We implemented the mapping procedure as Unity scripts. Developers of the virtual world may attach them to GameObjects via the Unity editor and enable the objects to function as a display. The VDS and the VDC are connected in a client-server fashion using Sockets. Thereby, the VDS acts as server, the VDC as client. For serialization purposes, we use Google's *ProtocolBuffer*[2] library. It offers a fast, convenient, and platform-independent serialization [10]. As ProtocolBuffer does not offer a stable C# library, we also use the *protobuf-net*[3] library for serialization within Unity. Furthermore, we implemented the protocol from Section V-D. For interaction between the application and the virtual display, the `OBJ` for showing new content and the `CTR` command for bi-directional interaction are relevant.

B. Implementation of Inter-User Communication

Besides the interaction of the application with virtual displays, our protocol supports inter-user communication. The communication uses the connection between the Communication Controller in the virtual world and the communication services in both worlds. In this section, we focus on how the communication is implemented. As the protocol supports three communication styles – text, audio, and gestures – the section is threefold. Unity offers a low level API for network transmissions in the *UNET* package. This API supports receiving and sending of binary data. Further, it supports the definition of channels. For the sake of a clear separation of message types and a corresponding logic to handle the messages, we define a channel for each message type, hence, each protocol command. Each channel can provide an individual quality of service (QoS) level. Table I shows the channels, their QoS levels, data types, and the corresponding protocol commands.

[2]ProtocolBuffer website: https://developers.google.com/protocol-buffers/
[3]protobuf-net website: https://github.com/mgravell/protobuf-net

TABLE I
CHANNELS WITH THEIR QUALITY OF SERVICE (QOS) LEVELS, DATA TYPES, AND CORRESPONDING PROTOCOL COMMANDS (CMD).

Channel	QoS	Data Type	Cmd
Chat	Reliable	`TextChatMessage`	`TEX`
Voice chat	Unreliable	`CompressedFile`	`AUD`
Gesture	Reliable	`GestureSelection`	`GES`
Group management	Reliable	`Group`	`GRP`
Control	Reliable	`StudentCom`	`ACC`

Text chat messages are implemented as transmitted text messages that are shown in a separate UI element within the virtual environment. We implemented the voice chat using the *Nspeex*[4] library. This library captures audio signals from microphones and splits it into decoded chunks. As frequency we use 16,000 Hertz. After transmission, we use the library to encode the information and return it via audio devices to users. For gesture-based communication, we defined a set of shortcuts. Each command reflects a gesture of the user's avatar which is triggered by the VDC using the Animation System of Unity. Additionally, Unity is responsible to manage groups of users and react accordingly on the activities for access control by specifying the corresponding rights for users.

As we are using the BASE middleware as pervasive middleware in the physical representation and the Unity game engine for representing the virtual environment, our implementation is specific to them. However, all implemented components are reusable by other applications. Furthermore, developers may extend our system with own virtual devices written for Unity or physical applications, hence, additional BASE services. Through the well-defined protocol, it is possible to exchange the BASE middleware with another middleware but still using the Unity game engine. For future work, we plan to implement our approach using other pervasive middlewares and game engines to improve the interoperability.

VII. EVALUATION

For evaluating the prototype of our approach, we implemented a virtual meeting room which is connected to a physical meeting room. Next, we introduce the scenario in detail and describe the evaluation setting. Afterwards, we present the evaluation result of the performance measurements of our prototype implementation.

A. Evaluation Setting

For the evaluation of our VDS, we implemented a virtual meeting room. The virtual environment is equipped with several displays. Figure 5 shows a part of the meeting room and some of the displays with different content. Besides a projector in the front and two large screens at the sides of the room, the room is equipped with several PCs in the back, which illustrates personal virtual devices of users. An application can access these displays using the BASE services separately and can send different content to each of them.

[4]Nspeex website: https://nspeex.codeplex.com/

As content types the virtual meeting room supports images and videos as well as PDFs for presentations. Also the full capability of the inter-user communication system is build into the virtual meeting room. Users can communicate with gestures, text, and audio messages and the presenter is able to group users together. Additionally, the presenter can control access to communication channels for users.

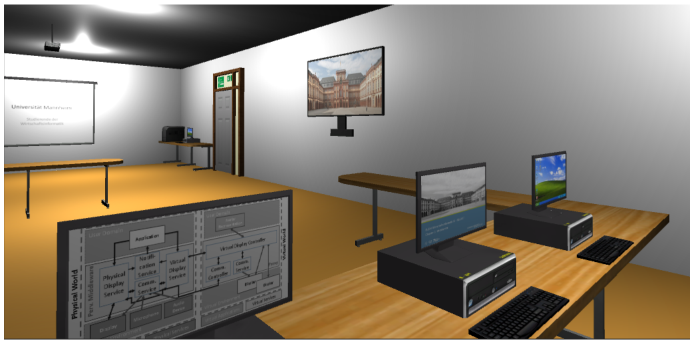

Fig. 5. Screenshot of the example virtual environment.

We implemented an application for the virtual meeting room, which uses the PDS and the VDS. This application allows to pick displays in the physical or virtual space and transmit selected content to them. The user of the application has full control over what content is shown on which display and can forward PDF presentations or start and stop videos.

TABLE II
COMPARISON OF THE EXAMPLE APPLICATION AND REQUIREMENTS.

Requirement	**Implemented Feature covering Requirement**
Content Type	Text, Image, Video, PDF
User/Sensor Input	Forward slides
Responsiveness	—
Communication	Text, Speech, Gesture
Access Control	Group management, mute users

As Table II shows, the implemented functionality already fulfills four out of the five requirements we derived in Section IV. For analyzing the responsiveness of our approach, we performed a quantitative evaluation of the VDS. First, we measure the time difference between sending an image to the PDS and the VDS. Next, we evaluate the time for transmission of an image to the virtual display. Last, we analyze how the amount of displays influences the time needed for sending.

Except for the last evaluation, the evaluations were executed on a PC with an Intel Core i7 4700MQ quadcore CPU with 2.40GHz, 16 gigabytes memory, and a Nvidia Geforce GT 730M graphics card. The system uses Windows 10 64bit, Java 8.0.101, and Unity3D version 5.3.4. We perform most of the measurements locally to measure only the computational overhead and neglect uncertainty introduced by varying transmission times via the network. The third evaluation is performed in a distributed setting.

B. Evaluation Results

To measure the time needed to show an image on the display we use three images with 1000x1000, 750x750, and 500x500 pixels. As Figure 5 shows, the virtual displays cover only a limited area of a physical display (up to a 4k resolution). Therefore, the resolutions for the pictures are large enough. We first measure the time needed for the transmission between the application and BASE service. The transaction can be divided into three parts: (i) transmission to the VDS or PDS, respectively, (ii) transmission from VDS to VDC, and (iii) time needed for displaying the picture on a virtual device. As the time needed to display the image on a physical display after it arrives at the PDS is less than 1ms on average, it is neglected. The evaluation is repeated 500 times per image. Figure 6 shows the time needed in total and for each of the three steps. Even for the largest image, the time difference between the appearance on the physical and virtual display is with 38.22ms not recognizable. This does not influence user experience as for users it seems as the physical and virtual displays are synchronized. In detail, the file transmission between the BASE service and VDC takes on average between 7.22ms for smallest image resolution and 7.89ms for the largest. The BASE middleware consumes up to 66% of the time for sending the image from the application to the service (59% on average). Thus, a middleware which is optimized for handling media content could improve the performance.

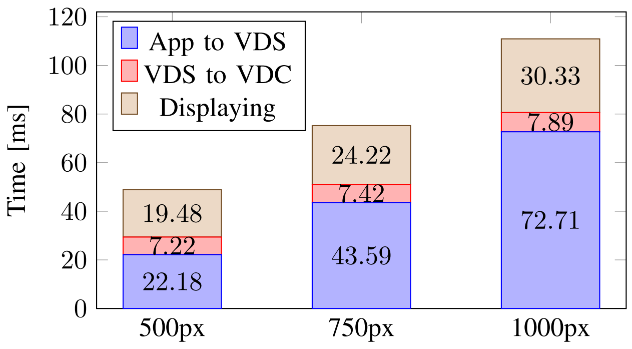

Fig. 6. Transaction times for showing a picture on a virtual display.

Next, we evaluate the time needed to send an image to several virtual displays. For this measurement, we use only the largest image with 1000x1000 pixels. The amount of virtual displays is increased up to eight displays placed in our virtual meeting room. We again measure the transmission time between the VDS and VDC and the time needed to display the images. Figure 7 shows the corresponding transaction times. The time for the complete process increases on average by 27.92ms for each additional display. Sending an image to all eight displays simultaneously needs on average 228.64ms. In an optimized scenario it is highly unlikely for all applications to update their virtual display at the same time. Additionally, there is no need to send the content several times if multiple displays should show the same content as the received file is saved at the hard disc and all displays can access it.

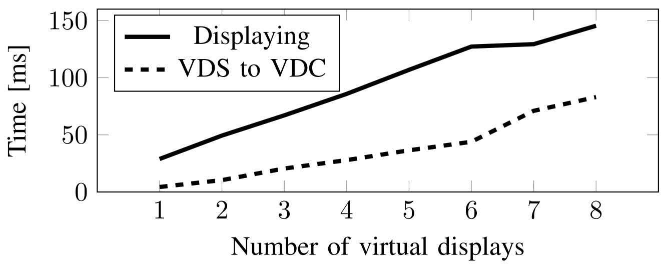

Fig. 7. Transaction times for showing pictures on up to eight virtual displays.

For the third quantitative evaluation, we measure the time for sending an image from one application to several VDSs over a network. Therefore, we started 16 instances of the VDS on 4 PCs. As soon as a VDS receives an image and is ready to relay it to the VDC, it sends an acknowledgment back to the application. We measure the duration between the start of the sending process until the reception of the acknowledgment. As Figure 8 shows, the time for sending an image increases linearly by 26.13ms for each additional receiving service. Hence, the service scales in a satisfactory manner to support the users of a virtual meeting room. For integrating additional users, it is also possible to split the distribution of content between services in a hierarchical manner.

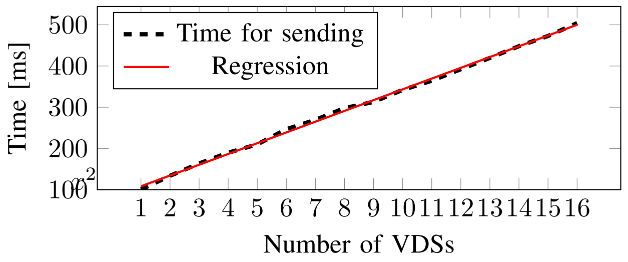

Fig. 8. Transaction time for an image from one application to several virtual displays services (VDSs) over a network with acknowledgment and regression function y = 26.129x + 82.118. $R^2 = 0.9987$ indicates an almost perfect fit of the observation with the regression.

The evaluation showed that the VDS meets most of our requirements derived in Section IV through the implemented functionality. We showed with three measurements that the prototype fulfills the requirement for responsiveness. The biggest bottleneck in providing new content to the virtual displays is the transmission of the data via the BASE middleware, which can be boost by using a media-centric middleware.

VIII. Conclusion

In this paper, we introduced our approach for transferring our Interactive Display Service presented in [1] to a virtual environment. The VDS is highly flexible regarding the use case and applications. Application developers get the opportunity to build a link between the distribution of their content in the physical and virtual environment. Remote users get the chance to have the same experience and learning or working environment as their colleagues. The service is designed to run on a high variety of devices, but the effect can be intensified by using specialized hardware, such as head-mounted displays.

We plan to refine the PDS and the VDS in further iterations. First, we plan to implement our approach using other pervasive middlewares and game engines. This step improves the interoperability and allow a higher variety of devices to join the service. Additionally, we plan to tackle performance issues with pervasive middlewares, which are better optimized for multimedia content. Furthermore, new services will be introduced to the middleware, to allow a better connectivity between the users and distribution of content. For instance, it would be possible to add a file service in the infrastructure of the pervasive environment to cache and distribute user content to different receiving devices, as well as rendering services to relief user and display devices with high workload dramatically. To reduce the network load, it would also be possible for the VDC to propagate the needed resolution for a virtual display, so that the rendering service can scale the content down before transmitting. It is also conceivable to fit the pervasive middleware with an easy to use configuration interface. This could enable presenters or lecturers to adjust the distribution of information or content in the environment and between output devices on the fly. This would enable to adjust the middleware to new use cases and scenarios without the input of an administrator or even developer. Additionally, a large scale evaluation with students during a lecture is planned to evaluate with a multitude of devices and content types as well as the usability from a lecture's point view.

Acknowledgment

We would like to thank our former students Christiane Lupui and Maria del Carmen Ocón Palma for their contribution.

References

[1] J. Naber, D. Schäfer, S. VanSyckel, and C. Becker, "Interactive display services for smart environments," in *Proc. PICOM*, 2015, pp. 2157–2164.

[2] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel, "BASE - a micro-broker-based middleware for pervasive computing," in *Proc. PerCom*, 2003, pp. 443–451.

[3] N. A. Streitz, J. Geissler, J. M. Haake, and J. Hol, "Dolphin: Integrated meeting support across local and remote desktop environments and liveboards," in *Proc. CSCW*, 1994, pp. 345–358.

[4] T. Riisgaard Hansen and J. E. Bardram, "Activetheatre: A collaborative, event-based capture and access system for the operating theatre," in *Proc. UbiComp*, 2005, pp. 375–392.

[5] P. C. Shih, D. H. Nguyen, S. H. Hirano, D. F. Redmiles, and G. R. Hayes, "Groupmind: Supporting idea generation through a collaborative mind-mapping tool," in *Proc. GROUP*, 2009, pp. 139–148.

[6] O. Hagsand, "Interactive multiuser VEs in the DIVE system," *IEEE MultiMedia*, vol. 3, no. 1, pp. 30–39, 1996.

[7] R. Lea and K. Honda, Yasuakiand Matsuda, "Virtual society: Collaboration in 3d spaces on the internet," *Computer Supported Cooperative Work (CSCW)*, pp. 227–250, 1997.

[8] A. De Lucia, R. Francese, I. Passero, and G. Tortora, "Development and evaluation of a virtual campus on second life: The case of secondDMI," *Computers & Education*, vol. 52, no. 1, pp. 220–233, 2009.

[9] P. Petridis, I. Dunwell, S. de Freitas, and D. Panzoli, "An engine selection methodology for high fidelity serious games," in *Proc. VS-GAMES*, 2010, pp. 27–34.

[10] J. Feng and J. Li, "Google protocol buffers research and application in online game," in *IEEE Conference Anthology*, 2013, pp. 1–4.