UNIVERSITÄT
MANNHEIM

Runtime Reconfiguration of physical and virtual Pervasive Systems

Inauguraldissertation

zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften
der Universität Mannheim

vorgelegt von

Jens Naber

aus Heidelberg

Dekan:	Prof. Dr. Christian Becker
Erstreferent:	Prof. Dr. Christian Becker
Zweitreferent:	Prof. Dr. Dirk Ifenthaler
Tag der Disputation:	17. Dezember 2019
Prüfungsausschuss:	Prof. Dr. Christian Becker (Vorsitzender)
	Prof. Dr. Dirk Ifenthaler

Abstract

Today, almost everyone comes in contact with smart environments during their everyday's life. Environments such as smart homes, smart offices, or pervasive classrooms contain a plethora of heterogeneous connected devices and provide diverse services to users. The main goal of such smart environments is to support users during their daily chores and simplify the interaction with the technology. Pervasive Middlewares can be used for a seamless communication between all available devices and by integrating them directly into the environment. Only a few years ago, a user entering a meeting room had to set up, for example, the projector and connect a computer manually or teachers had to distribute files via mail. With the rise of smart environments these tasks can be automated by the system, e.g., upon entering a room, the smartphone automatically connects to a display and the presentation starts. Besides all the advantages of smart environments, they also bring up two major problems. First, while the built-in automatic adaptation of many smart environments is often able to adjust the system in a helpful way, there are situations where the user has something different in mind. In such cases, it can be challenging for unexperienced users to configure the system to their needs. Second, while users are getting increasingly mobile, they still want to use the systems they are accustomed to. As an example, an employee on a business trip wants to join a meeting taking place in a smart meeting room. Thus, smart environments need to be accessible remotely and should provide all users with the same functionalities and user experience.

For these reasons, this thesis presents the PerFlow system consisting of three parts. First, the PerFlow Middleware which allows the reconfiguration of a pervasive system during runtime. Second, with the PerFlow Tool unexperienced end users are able to create new configurations without having previous knowledge in programming distributed systems. Therefore, a specialized visual scripting language is designed, which allows the creation of rules for the communication between different devices. Third, to offer remote participants the same user experience, the PerFlow Virtual Extension allows the implementation of pervasive applications for virtual environments. After introducing the design for the PerFlow system, the implementation details and an evaluation of the developed prototype is outlined. The evaluation discusses the usability of the system in a real world scenario and the performance implications of the middleware evaluated in our own pervasive learning environment, the PerLE testbed. Further, a two stage user study is introduced to analyze the ease of use and the usefulness of the visual scripting tool.

Acknowledgments

This thesis would not have been possible without the support and help of several people, many of whom may not even be aware of their contribution.

First of all, I would like to thank my supervisor Prof. Dr. Christian Becker for all his support and mentoring. Christian, thank you for giving me the chance to join your research group, after I one day suddenly appeared at the doorstep of your office. I am also very grateful that you always have an open door and ear for us, independent of the topics we are approaching you with. Furthermore, it was always a pleasure traveling with you, experiencing interesting conferences and nice food and wine. No matter if the destination was Hong Kong or Cottbus.

I would like to thank Prof. Dr. Dirk Ifenthaler for his willingness to act as the second supervisor and join the board of examiners. Especially, considering approaching him quite last-minute with a tighter schedule than initially planned.

I would like to thank all the people I worked with at the chair, namely Dr. Patricia Arias-Cabarcos, Martin Breitbach, Melanie Brinkschulte, Dr. Janick Edinger, Kerstin Goldner, Melanie Heck, Benedikt Kirpes, Sonja Klingert, Dr. Christian Krupitzer, Markus Latz, Yugo Nakamura, Martin Pfannemüller, Dr. Vaskar Raychoudhury, Dr. Felix Maximilian Roth, Dr. Dominik Schäfer, Dr. Sebastian VanSyckel, and Anton Wachner. Because of you all it was always a pleasure to come to the office and it felt more like meeting friends than going to work. The support and team spirit in this group, especially during stressful and long days before deadlines, can not be described. In particular I would like to thank Christian, who introduced me to a new and exciting research project and at the same time to a professional soccer club. Thank you Dominik for always having great advice, when I showed up in your door. Janick, thank you for always having an open ear and helping out wherever you can, no matter how stressful your own workload is. Thanks to Max for being an outstandig office neighbour over the last six years and on several floors. Further, thanks to Martin and Martin for your support and helping hand during the final phase of writing the thesis.

Also, I would like to thank all the people I had the pleasure of working with at the Mannheim Business School. Thanks to Prof. Dr. Jens Wüstemann, Dr. Ingo Bayer, and Dr. Florian Heger for giving me the chance to work at the MBS. Further, thank you Florian for providing me with interesting projects and for your understanding regarding the coordination of work and thesis. Thank you Nina for always asking about the status of my thesis and the mental support during these conversations. Also, thanks to Bärbel, Björn, Malte, and Lisa for your help when needed and the talks we had every time I visited you at your office.

I would also like to thank each student and research assistant involved in the project in some way for their contributions, especially, Lena Burger, Xaver Fleischer, Halil Gayretli, Alexander Glatt, Christina Kannegießer, Dennis Knödler, Christiane Lupui, Maria del Carmen Ocón Palma, Steffen Schmitz, Roman Scholz, Nadja Seemann, Kai-Arne Theis, Irina Toncheva, Alexander Tsarov, and Anne Wesch.

Last but not least, I would like to thank my family and friends for always being there for me. To my parents, Martina and Jürgen, thank for supporting me on my way and enabling me to reach this achievement. To my sisters, Lisa and Lena, for being such outstanding sisters. To my grandparents, Alfred, Ilse, Theo, and Edith, for encouraging me and always crossing their fingers. To my girlfriend, Sonja, thank you for supporting and motivating me. You had my back and a lot of patience during the stressful phases, especially the last few months.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

3D	three dimensional
API	Application Programming Interface
CRUD	Create, Read, Update, and Delete
DVE	Distributed Virtual Environments
GPS	Global Positioning System
HTML	Hypertext Markup Language
ID	Identifier
IDE	Integrated Development Environment
IoT	Internet of Things
IT	Information Technology
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
OS	operating system
PDA	Personal Digital Assistant
PDF	Portable Document Format
PNG	Portable Network Graphics
REST	Representational State Transfer
RPC	Remote Procedure Call
SAX	Simple API for XML
SDK	Software Development Kit
SLoC	source Lines of Code
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
TAM	Technology Acceptance Model
TCP	Transmission Control Protocol
UI	user interface
UTAUT	Unified Theory of Acceptance and Use of Technology
UWP	Universal Windows Platform
VoIP	Voice over IP
VR	Virtual Reality
XML	Extensible Markup Language

1. Introduction

In the recent years, pervasive computing outgrew the pure research and found its way into everyday's life of users in the form of smart environment such as smart homes, offices, or classrooms. With this step forward the target audience significantly increased and developers can no longer expect all users to have detailed knowledge about the technicalities of pervasive middlewares. Thus, it is getting increasingly important also for users without a background in information technology to being able to influence the behavior of such smart environments and customize them for their use case. So that for instance also a history teacher without motivation to familiarize with the system is able to give students access to the projector or distribute additional learning materials. A further challenge is the increased mobility of users, leading to the urge to access these smart systems remotely.

The goal of this thesis is to tackle those challenges in two different ways. First, we design a configurable pervasive middleware enabling developers to connect their applications and users to configure how these applications should communicate with each other. To also enable end users without further knowledge to configure the pervasive system, the middleware includes a visual scripting tool. Second, a virtual extension of the pervasive middleware allows developers to migrate their applications to a virtual environment and thus providing remote users with a similar experience to those in the physical pervasive environment.

This chapter illustrates the motivation for the thesis and defines the problems at hand. Afterwards, the research questions are discussed before giving an overview of the scientific contributions and the structure of the thesis.

1.1. Motivation and Problem Definition

Pervasive computing allows the integration of smart and connected devices into everyday objects and environments with the goal of supporting users in their daily routines. In recent years, we observed a steady increase of smart objects in different forms. This includes wearables (e.g., smart watches or fitness trackers), personal devices (e.g., smartphones, tablets, or computers), and stationary devices (e.g., smart TVs or speaker systems). Not all devices are that obvious to the user. Instead, many are completely integrated in the environment and therefore almost invisible such as light or proximity sensors, smartdoor bells, and heating or air conditioning controls. Seen on their own, all these devices already provide benefits to their users. By combining their capabilities it is possible to increase the usefulness significantly. This enables scenarios where on the way home a user's smartphone tells the heater at home to turn on or the smartdoor bell shows who is ringing on the TV nearest to the user. With the help of pervasive middlewares it is possible to enable communication between the smart devices and give developers the possibility to utilize them for their applications.

By combining all devices in a smart environment to one pervasive system, it is possible to increase the performance and experience of users significantly. This is especially true in multiuser working environments such as offices, meeting rooms, or classrooms. Lectures in a smart classroom can be enriched with multimedia content and the lecturing style is getting more dynamic including e.g., classic head-on lectures, group work, or student presentations. To achieve this, teachers need the possibility to show content on projectors or screens, exchange files with students, or enable collaborative work on assignments. Only a few years ago, this required a lot of manual effort, as many devices were controlled separately. This included inserting DVDs into DVD players, connecting laptops to portable projectors, or giving students access to workstations. In addition, users needed to make sure that these devices were compatible. Today, smart environments allow for an easy integration of these features and offer automatisms designed to support the user during everyday tasks. Thus, systems can be configured to automatically recognize the laptop of the teacher and start a presentation on the large screen or to prepare the room by closing the shutters and turning on the heater based on a scheduled meeting in the calender. These possibilities led to a fast pace adoption and today many users are able to benefit from smart environments.

However, the advantages offered by these smart environments and the widespread availability come with two major caveats. First, not all users encountering such a pervasive system are experienced with smart devices and tech-savvy enough to immediately understand how to use them and get used to them. While it would be possible to acquire the needed knowledge, many people might not willing to invest time and effort into studying new technologies if they are not interested in the technology from the beginning. Automatic configuration of context aware pervasive systems and predefined setups by administrators can help to take the responsibility off the end user. Nonetheless, there are, circumstances in highly dynamic scenarios such as a lecture, where the user needs to be able to adapt the system without reaching out for help. As also discussed by Holloway and Julien [82], there is a need for end user empowerment in pervasive environments. Thus, one major challenge is to enable end users without deeper knowledge of pervasive systems to being able to influence the configuration of the system.

The second challenge we see in the realization of smart multiuser environments is the increased mobility of users. While the inclusion of pervasive systems in environments such as classrooms or meeting rooms positively effects the user experience and offers many possibilities for collaborative work, it only benefits users that are on location. However, in such use cases we often encounter many mobile users such as students abroad who want to attend a lecture at their home university or employees on a business trip required to join a meeting. These users still need access to the functionalities offered by the system remotely and preferably want the same experience as the users physically on location.

1.2. Research Questions

Based on the motivation and the discussed challenges, the objective of this thesis is to enable end users without further knowledge of pervasive computing to configure pervasive systems for their current use case. Further, users should be able to access the system remotely without lacking behind in user experience. Therefore, a pervasive middleware is needed that gives developers the possibility to define what information their applications are able to send or receive and that allows for reconfiguration during runtime. Additionally, this should be possible without interfering with the user experience and the current task that the pervasive system is used for. This leads to the first research question:

How to configure and control the pervasive system during runtime without compromising the user experience?

Even if the pervasive middleware can be reconfigured during runtime, this reconfiguration should also be possible for users without training or prior knowledge. Thus, the system has to offer a simple to use and fast to learn interface to enable end users to create configurations on their own, leading to the second research question:

How can visual scripting be used to enable unexperienced and untrained users to change the configuration of a pervasive system?

In addition to the reconfigurability of the pervasive system, the objective is also to integrate remote users. Therefore, the pervasive middleware has to offer developers the possibility to make their applications and services accessible from outside the smart environment. Further, the system should allow remote users to have the same user experience as their local peers. This leads to the third and final research question:

How can remote users access and use pervasive systems with a comparable experience to users physically on location?

1.3. Contributions

In the course of this thesis, we will present PerFlow, a configurable pervasive middleware with a virtual extension for remote users. The thesis comprises the design, implementation of a prototype, and an extensive evaluation. The five main contributions are as follows:

First, the state of the art is analyzed by conducting a literature review. Therefore, we developed two classifications for configurable pervasive middlewares and remote participation systems. These classifications are in the next step used to categorize existing approaches and identify the research gap. Further, the requirements for such a middleware are derived with the help of a pervasive classroom scenario.

Second, the design for a configurable pervasive middleware, called PerFlow Middleware, is introduced, allowing developers to define what information their applications are able to send. The middleware then allows the reconfiguration of the system during runtime. It is further responsible for access control, distributing the configuration, and controlling the communication between applications.

Third, we extended the middleware with the PerFlow Tool, a visual scripting tool for creating the configurations. Therefore, we introduce an easy to use visual scripting language, allowing end users to reconfigure the system for their needs. With the help of tool users are able to create rules for the information flow in the system via drag and drop. Afterwards, the new configurations are handed over to the middleware.

Fourth, we present the PerFlow Virtual Extension, a virtual extension allowing remote users to access smart environments. This extension utilizes an existing game engine to enable the creation of three dimensional virtual environments. Developers are enabled to access all functionalities of the PerFlow Middleware from within the game engine to migrate their services and applications to the virtual environment.

Finally, we implemented the complete PerFlow system with all extensions as a prototype and extensively evaluated. The evaluation is split into four parts, starting with a proof of concept with PerLE, a testbed for a pervasive learning environment. Further, we analyze the implementation effort for developers and perform several performance measurements for the middleware. Lastly, a two stage user study is conducted for the PerFlow Tool.

1.4. Structure

The remainder of this thesis is structured as follows. After the introduction, Chapter 2 covers fundamental knowledge about pervasive computing, visual scripting, and virtual environments. Following, Chapter 3 presents and discusses different approaches for configurable pervasive middlewares and remote participation systems. In Chapter 4, we first give a concrete scenario for our system and afterwards use this scenario to derive our requirements. Chapter 5 describes the system model and the terminology used for our design. In Chapter 6, we introduce the design of our pervasive middleware. Therefore, we first take a look on how the middleware allows the reconfiguration of the system before discussing the visual scripting approach for end users and the virtual extension for remote participants. Afterwards, Chapter 7 presents the implementation of the prototype that we evaluate in Chapter 8 for an extensive evaluation. Finally, Chapter 9 concludes the thesis and gives an outlook to future research objectives.

2. Theoretical Foundations

The following chapter provides background information required in the remainder of this thesis. Therefore, we will discuss fundamental technologies used for the development of the PerFlow system. First, pervasive systems are discussed in Section 2.1 together with pervasive computing, service oriented architecture, and adaptation of pervasive systems. These technologies are the foundation of the later introduced PerFlow Middleware. Second, Section 2.2 introduces visual scripting, which can be used to empower end-users without Information Technology (IT) background. Visual scripting is therefore the ideal basis for the PerFlow Tool, to enable users to influence the behavior of a pervasive system. Lastly, in Section 2.3 we take a look on virtual environments and how they are developed. The high immersion of a virtual environment can be used to transfer the physical experience to remote users.

2.1. Pervasive Systems

With the increasing number of devices present in today's environment, the need to enable communication between them is getting higher. Developers face the challenge of connecting the heterogeneous devices and simultaneously providing a seamless experience for the user. This is the driving force behind the emergence of pervasive computing. The following section will first discuss the concept of pervasive computing before explaining service oriented architectures as one of the most popular ways of realizing pervasive systems. Lastly, different concepts of adapting the system to the needs of the user and the surrounding environment are discussed.

2.1.1. Pervasive Computing

Distributed and mobile computing are the foundation of pervasive computing [169]. While distributed computing allows connected devices to share their functionalities and thus increase the potential, mobile computing integrates mobile devices and enables the communication between them [167]. Pervasive computing is the combination of both approaches and allows the access to information and functionalities “all the time everywhere” [167]. The mobility of users, and thus devices, is one of the biggest challenges of pervasive computing [167,169]. Therefore, the systems need to be able to cope with devices moving within the environment, but more importantly also joining and leaving the environment at any time.

Mark Weiser envisioned already in 1991 [197, 198] the transition from desktop computing to physical environments equipped with connected computationally enabled devices dedicated to specific tasks. Further, these devices should be integrated seamlessly into the environment and aid the user in fulfilling their everyday tasks, and thus eventually reaching an ubiquitous state. Satyanarayanan is therefore also talking about “technology that disappears” [169]. According to Norman [140], the related concept of *information appliances* was already described in 1978 by Jef Raskin in internal documents at Apple. These anticipated the shift from all-purpose computers to devices developed to fulfill specific tasks as good as possible, e.g., digital cameras, smartphones, or e-book readers.

The realization of the pervasive computing principles for a given scenario is typically called a pervasive system [115]. These systems combine the users and devices present in a physical environment, where the users interact with the devices either deliberately or not. The devices can occur in the form of mobile devices, computers, or smart devices [169]. To allow developers to implement applications and services for such a pervasive system, they are typically assisted by a pervasive middleware (also called pervasive platform) [49]. Such a middleware handles the connection and communication between devices and offers developers services for the discovery of devices and functionalities in the system. Today, many different middlewares exist for the development of pervasive systems, fitting to different needs and possible use cases, e.g., Gaia [35, 162], Aura [57, 180], or CORTEX [192]. A more detailed survey describing and comparing the different pervasive middlewares was conducted by Raychoudhury *et al.* [159].

Following on pervasive computing, the next large advancement was the Internet of Things (IoT), where not only computational devices but also physical “things” are able to communicate and share information. IoT has the possibility to “connect the world’s objects in both a sensory and intelligent manner through combining technological developments in item identification (“tagging things”), sensors and wireless sensor networks (“feeling things”), embedded systems (“thinking things”) and nanotechnology (“shrinking things”)” [183]. Kevin Ashton, who coined the phrase “Internet of Things (IoT)” in 1999, said that “we need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory” [11]. The concept brings pervasive computing to an even larger scale by globally connecting the different devices surrounding users everyday [125]. Predictions talk about 24 billion interconnected devices until the year 2020 [69]. Users are potentially surrounded by devices located directly in their environments. This does not only include obvious examples like smartphones or computers, but also more subtle devices, e.g., room lighting, cars, or refrigerators [12].

With the emergence of more and more computationally enabled devices, pervasive computing and IoT have a growing importance. Today, not only academia is showing interest, but also industry and private users are recognizing the possibilities. The use cases and application areas of pervasive computing and IoT are manifold and include smart home [77], logistics [117], agriculture [195], smart office spaces [4], and healthcare [154, 188].

2.1.2. Service-Oriented Architectures

Service-Oriented Architecture (SOA) [152] is widely used in pervasive computing [158] and IoT [12]. Many existing pervasive middlewares are based on the principle of SOA, e.g., BASE [16], iPOJO [50], and DigiHome [163]. In a SOA, services are used as the modular and reusable basic elements, which encapsulate specific functionalities offered to applications [152]. This could for instance be a thermostat, which offers the functionality to control the heating, or a thermometer, which gives access to the ambient temperature. Service-Oriented Computing (SOC) offers a dynamic and flexible way of developing applications [88] to combine the functionalities of different services in the system. In the example, an application could access the service of the thermometer to receive the current temperature, and dependent on that, trigger the thermostat. Additionally, applications are able to look for and access services during runtime [153], allowing to react dynamically to their surrounding. In our example, the application could for instance read the temperature and if it is too low, search for a heater and vice versa search for an air conditioning unit if it is too high. As the service encapsulates the functionality from the application [152], it is also easily possible to adjust the service without the need to alter the application.

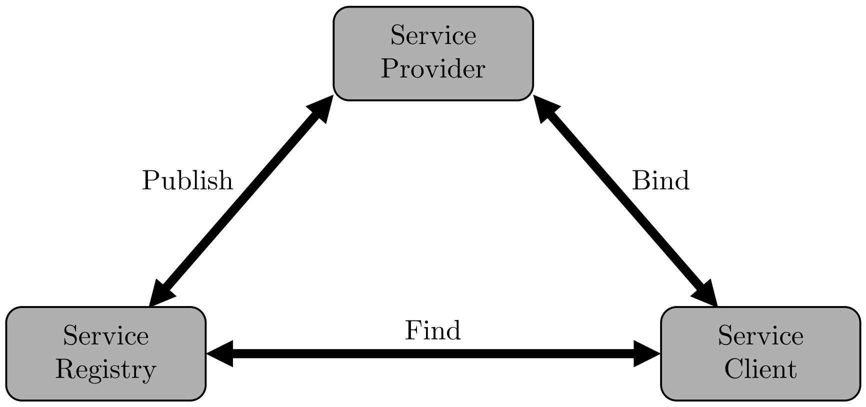


Figure 2.1.: Logical view on the Service-Oriented Architecture by Papazoglou [152] describing the interaction between Service Registry, Provider, and Client.

The SOA defines three different elements and the relationship between them [152], which can be seen in Figure 2.1. The *service provider* [152] offers either access to information or functionalities. The *service client* [152] (often also called *service consumer* [22]) on the other hand wants to utilize the access. The last element is the *service registry* [152] (also called *service broker* [50]) which stores the description of all available services and offers them to possible clients. If a service provider joins the system, it publishes a description of the offered service at the service registry. This description includes information about the offered functionalities of the service, but it may also contain additional information, e.g., regarding the location or availability of the service. In the next step, an application acting as a service client may ask the service registry for possible services fitting the needed functionality. This request may also contain additional requirements, e.g., asking for a service at a specific location. The service registry then returns the descriptions fitting to the request from the client. After the service client has received the descriptions, it can directly access the service of the corresponding provider.

The process starting with the advertisement of the service by a provider until the client is able to access it, is called *service discovery*. Many different protocols supporting this service discovery process exist, e.g., INS [2], DEAPspace [139], or UPnP [95]. Zhu *et al.* [201] compare a multitude of protocols regarding different offered functionalities and interaction strategies. In total, they are comparing 18 different dimensions including communication method (unicast, multicast, or broadcast), service selection (automatic or manual), service status inquiry (notification or polling), and discovery scope (network topology, user role, or context). This survey shows, that while many different approaches for service discovery exist, a deliberate choice considering the planned use case is required.

2.1.3. Adaptation

Pervasive Systems are often very dynamic regarding their use case and the participating devices. The tasks users want the system to solve may vary vastly during the runtime of a system for instance, a lecture in a pervasive classroom may start with a presentation by the lecturer and turn into group work, where the students need controlled access to the system. Additionally, while some devices may be stationary, e.g., the projector in the classroom, many devices are able to join and leave the system at any time, e.g., the smartphones of students or the notebook of a presenter. Thus, it is important that the pervasive system is able to adapt to its environment and "to the dynamics entailed in human behavior" [78].

Adaptation can happen at three different levels of the pervasive system. Herrmann *et al.* [78] and Satyanarayanan [170] differentiate between adapting the user's behavior, the pervasive application, and the environment. The highest impact on the user would be asking to change his behavior. This could be for instance asking to move more slowly in order to stay within network range [78]. Adapting the pervasive application or environment is in the best case unnoticed by the user, even though it may influence the service offered to him. This could be for instance either asking the application to reduce the resolution for an easier image transcoding or to increase the available resources in the environment to lower the bottleneck [170].

Further, it is of interest who is responsible to make the decision for an adaptation. This ranges from manual adaptation controlled by the user to completely automatic adaptation by the pervasive system [168]. For the manual adaption two different approaches exist. First, providing information about the system and possible adaptations, but leaving the actual decision to the user [43]. Second, giving the user the possibility to give preferences or select predefined adaptation routines, but the actual adaptation is decided by the system [15]. For a completely automatic adaptation, it is distinguished between reacting to a change in the context of the system [172], and proactively anticipating the change and with this adapting the system beforehand [186]. Proactive computing is also seen as the next step away from interactive computing, as the system is able to achieve its tasks unsupervised and without human interaction [194].

To enable any kind of adaptation, the pervasive system has to provide some degree of context-awareness [171]. Detecting changes in the context of the system is the basis for all decisions, no matter if the adaptation is manual, reactive, or proactive. According to Dey and Abowd, "context is any information that can be used to characterize the situation of an entity" [1]. The context of a system can be divided into further information about the user or the environment the system is placed in. The user related context includes the location of the user [23], the identity [166], or even the social environment [173] or emotional state and attention [43]. Regarding the environment of the system, the context may include conditions like time (time of day, season), temperature [23], lighting [173], and information about the infrastructure [173], e.g., available networks or devices.

2.2. Visual Scripting

Visual programming or scripting enables an increasing number of people solve complex tasks they would normally not be able to achieve without learning a programming language. Today, many different approaches exist for cases like education [116, 160], IoT [98], game development [70], or video processing [185]. One main goal of visual scripting is to provide users with "languages and environments that are more natural, or closer to the way people think about their tasks" [130]. This idea corresponds with traditional scripting as a higher level of programming, which enables a more rapid development and the combination of different applications [149]. Visual scripting is able to provide these benefits by introducing visual languages as high level programming with graphical elements. Myers *et al.* define visual scripting in its broadest sense as "any system that allows the user to specify a program in a two (or more) dimensional fashion" [128]. This does not include textual languages, as they are viewed as a one-dimensional stream, but this definition provides the basis for a multitude of different visual alphabets and languages. A more visual approach is also helpful for the users in other computer science areas, as the taxonomy introduced by Singh and Chignell [176] shows in Figure 2.2. Apart from visual language systems, this taxonomy also includes visualization for programs, data, and algorithms and graphical interaction systems. Thus, even programmers working with textual programming languages are able to simplify their workflow by for instance using data visualization to grasp the content of a database.

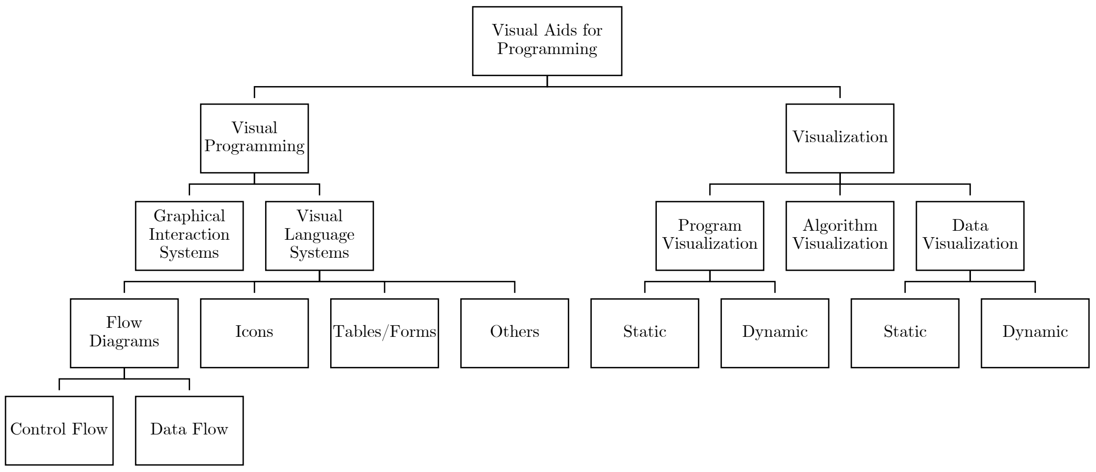


Figure 2.2.: Taxonomy of visual aids for programming by Singh and Chignell [176] differentiating between different approaches for visual programming and the visualization of programs, algorithms, and data.

According to Chang [31], a visual language is defined by "a set of visual sentences constructed with a given syntax and semantics". Thus, for such a language an alphabet and the possibility to compose valid rules is needed. Following, we will take a more thorough look on both of these components.

2.2.1. Visual Alphabet

While a traditional programming language is composed of a set of strings, a visual language can be seen as a set of pictures [59]. Golin and Reiss also defined that "a picture element is a primitive graphical object such as a line, shape or text string" [59]. The vocabulary of the visual language (the visual alphabet), according to Bardohl [14], needs to introduce a type set for the symbols and links of the language. These definitions allow for a multitude of different styles and depictions for visual alphabets and developers are free to choose or invent one fitting best to their desired use case.

Myers provided a taxonomy of 14 different styles of visual languages and categorized over 40 languages [129]. The different styles include flowcharts (e.g., OPAL [127]), petri nets (e.g., VERDI [63]), data flow graphs (e.g., HI-VISUAL [81]), iconic sentences (e.g., SIL-ICON [32]), and jigsaw puzzle pieces (e.g. Proc-BLOX [58]). Singh and Chignell [176] narrowed it down and included in their taxonomy seen in Figure 2.2 flow diagrams, icons, and tables/forms as the most common representations of visual alphabets.

According to Böhm and Jacopini [19], flow diagrams were developed without a systematic theory behind them and introduced in several papers with different purposes. Nonetheless, flow diagrams are seen as "suitable for representing programs, computers, Turing machines, etc." [19]. In their basic form flow diagrams consist of boxes and lines. Boxes are either functional, representing actual operations, or predictive, representing a decision on which the next operation should be. The lines are used to connect the boxes and to describe the transition between them. As the name already suggests, flow diagrams are a natural way to describe control or data flows.

In icon based alphabets the design of graphical symbols and interconnections can be more complex and it is possible to tailor the visual representation to the use case, e.g., the icon for 'input' could be a camera in a video editing program and a microphone in an audio recording program. Iconic languages are therefore context specific and "an icon image is chosen to relate to the idea or action either by resemblance (picture), or by analogy (symbol), or by being selected from a previously defined and learned group of arbitrarily designed images (sign)" [31]. If the user is familiar with the context of the visual language (e.g., video editing), the iconic alphabet may provide a quick recognition of the elements and a dense representation of complex content [42]. Untrained users on the other hand may have a hard time familiarizing with the icons and need training to use the visual language efficiently.

The last category requires the user to fill in tables or forms to define the behavior of the program. These are often used in database applications to enable the user to create queries or in spreadsheets. While forms and tables may be an easy and efficient way to communicate with users [176], they are comparably inflexible and often require deeper knowledge of the application (e.g., the structure of the database) from the user.

2.2.2. Visual Grammar

While the visual alphabets provide the basic building blocks of a visual language, they also have to be combined to form sentences. Therefore, it is important to define a visual grammar to enable the creation of syntactically correct sentences from the visual alphabet. A grammar consists of a finite set of rules [14] defining how the visual language has to be interpreted.

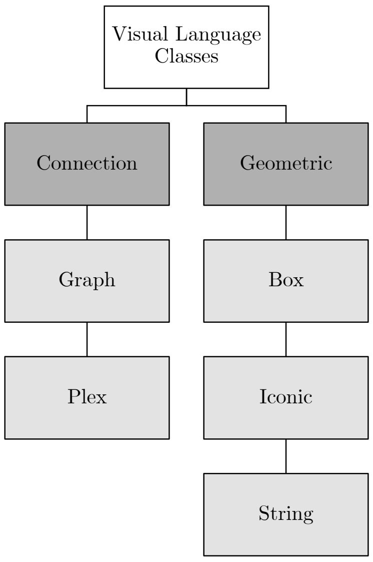
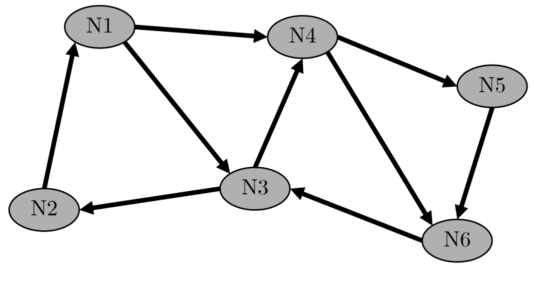
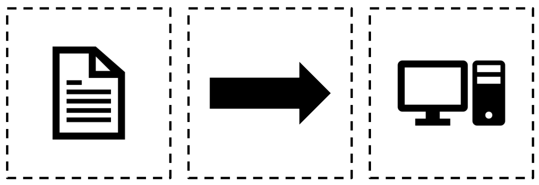


Figure 2.3.: Taxonomy of Costagliola *et al.* [38] for different classes of visual languages. The taxonomy differentiates between connection and geometric based as the two main classes of visual grammars.

In 2002, Costagliola *et al.* [38] developed a framework for the design of visual languages. In the process they introduced two main classes of visual languages. These classes are *connection* and *geometric based* visual languages, as can be seen in Figure 2.3, based on their visual grammar and how they compose their sentences. Connection based visual languages on the one hand are defined as a set of interconnected visual elements. The syntax of the language is given by the connection of the elements, while the relative position to each other is not influencing the outcome. Costagliola *et al.* define the two subclasses of *graph* and *plex*, while a plex is similar to a graph with the limitation, that it only can have a fixed number of connections [38]. On the other hand, in geometric visual languages the syntax is defined by the relative position of the elements to each other. In this case Costagliola *et al.* differ between the subclasses *string*, *iconic*, and *box*. For string based visual languages, the syntax is given by one integer describing the position of the element within a string of visual elements. The second subclass, the iconic languages, define their syntax as the relative position described by a pair of integers as their coordinate in the cartesian plane. Lastly, box based languages are defined by two coordinates for the upper-left and lower-right corner of the box. Therefore, it is possible to not only use the relative position of the boxes to each other, but also to use the size of the boxes to determine if one box overlaps an other one or is contained in it. In Figure 2.4, examples for a connection based graph and for a geometric based iconic sentence are given.



(a) Example graph



(b) Example iconic sentence

Figure 2.4.: Two examples for visual grammars based on graphs and iconic sentences by Costagliola *et al.* [38]. While in the case of a graph based language the connection between the elements is important for the syntax an iconic sentence uses the relative position of the icons to each other.

2.3. Virtual Environments

An environment is defined by three parts: content, geometry, and dynamics [48]. In the case of virtual environments, computer graphics are used to generate these features. The term *computer graphics* was first coined in the 1960s at Boeing, according to William Fetter by Verne Hudson, even though the basic principle of generating shapes or images through computers has already been used beforehand [54]. They were the first to generate human figures with computer graphics to determine the efficient layout of airplanes [55, p. 103]. These early attempts were realized by using wire-frame graphics. With the emergence of new rendering techniques like texture mapping [28], shadow mapping [200], tone mapping [187], subsurface scattering [73], and ambient occlusion [124], computer graphics made large steps towards photo realism in the following decades.

In the following, we will first look at augmented and virtual reality as a way to increase the immersion of virtual environments. Secondly, we discuss Distributed Virtual Environments (DVE), which allow to connect users within their virtual environments. Lastly, we talk about game engines as common middlewares to ease the development of virtual environments.

2.3.1. Augmented and Virtual Reality

How engaging and close to the reality a virtual environment comes is often described in the terms of immersion and presence. According to Slater *et al.* [178], immersion is an objective description of what a system provides to engage the user, e.g., surrounding displays, motion sensors, or convincing visuals. Presence on the other hand describes “a state of consciousness, the (psychological) sense of being in the virtual environment, and corresponding modes of behavior” [178]. Thus, in combination with other factors, like coherent world or believable narrative, the immersion of the system influences the perceived presence of the user strongly.

Augmented and virtual reality are the next logical step to increase the immersion of virtual environments. Benford *et al.* [18] provided the classification shown in Figure 2.5, illustrating that both augmented and virtual reality are highly artificial. The main difference is, that augmented reality still takes place in the physical world, while virtual reality transports the user completely into the

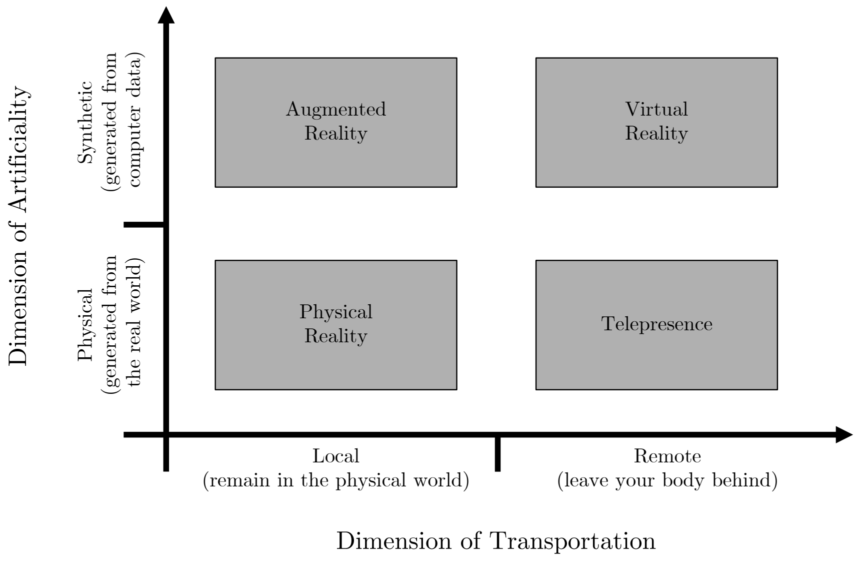

Dimension of Artificiality
Synthetic (generated from computer data)
Physical (generated from the real world)
Augmented Reality
Virtual Reality
Physical Reality
Telepresence
Local (remain in the physical world)
Remote (leave your body behind)
Dimension of Transportation

Figure 2.5.: A classification of mixed reality systems by Benford *et al.* [18] differentiating between the physical location of the user in relation to the content and artificiality of the viewed content.

virtual environment. Augmented reality overlays the physical environment of the user with computer generated graphical elements, while virtual reality leaves the real world completely out [18]. The idea and basic technology behind virtual reality is already several decades old and one of the first realizations was the head mounted display of Ivan Sutherland in 1968 [184]. Because of the high costs for specialized devices the computational power needed to simulate a virtual reality in real time, it took decades to establish the technology outside of academia or financially powerful sectors like the military or aviation companies. Only in recent years, computing devices got powerful enough to drive newly developed and cheaper virtual reality devices such as the Oculus Rift [51] or HTC Vive [85]. Augmented reality even arrived on devices with lower computational capabilities like smartphones [103].

2.3.2. Distributed Virtual Environments

As the scope of virtual environments increased steadily and the created world got larger and more complex, developers started looking into distributing the effort over networks. In the beginning, the idea of distributed virtual environments included the offloading of computational or storage heavy tasks such as computer-generated actors (artificial intelligence), world simulation, or storing the world state [182]. While these tasks may still be offloaded to increase the performance, the focus of the research shifted towards multi-user virtual environments [56]. The main goal is to allow a possible large group of participants to meet in a virtual environment and interact with each other. Therefore, the system has to be interactive and provide a persistent and consistent world to all users. Additionally, it needs to be able to scale with the amount of people using it. While in the beginning many distributed approaches relied on a traditional client-server architecture, many of them are based on a cloud architecture nowadays and research is looking into peer-to-peer architectures in order to use the devices of the participants for sharing the computational load [161].

Today, many distributed virtual environments, especially in the gaming sector, have several million active users, e.g., World of Warcraft[1]. While those virtual environments may distribute the users on several instances and different servers upon login to share the load, each instance nonetheless has to handle several thousand users simultaneously. This poses a great challenge for the developers, as the users are able to interact with the environment and the generated update events have to be sent to all other users in order to ensure a persistent and consistent world. To reduce the computational and network load of such events, two main questions need to be asked: Who needs to know about the event? And how often does he need an update about it? To answer these questions, the concept of interest management is introduced [126]. Here, the system tries to determine, which users are interested in the occurrence of an event, often based on the relative location of the user to the event. Different approaches are based on auras around users [17] (in the case of HyperVerse users and events [21]), spatial publish subscribe [87], or Voronoi-based Overlay Networks [86]. To increase the interval for sending events, the changes between two events can be interpolated with approaches like Dead Reckoning [151] or Continuous Events [76].

[1]`https://worldofwarcraft.com`

2.3.3. Game Engines

The video game industry is one of the major innovators behind the development of virtual environments. Statista[2], a market research company analyzing about 170 industry sectors, reported a 123.54 billion dollar revenue for the global video game market in 2018[3]. Further, they tracked a total of 9050[4] newly released games on Steam[5], the largest online distribution platform for computer games. These numbers show the high effort companies are investing in the development of video games and the gain they are able to generate out of it. During the last decades, game development studios not only invested heavily in the innovation of new techniques in areas like rendering, artificial intelligence, or online services, but also streamlined the development processes. This led to the emergence of game engines, middlewares specialized for the development of video games. Game engines contain different modules handling the input, output, and simulation of physics and game worlds [111]. According to Lewis and Jacobson, "the cost of developing ever more realistic simulations has grown so huge that even game developers can no longer rely on recouping their entire investment from a single game" [111]. Therefore, the development of game engines can help to spread the investment over several games. Many game engines are first developed for a specific game and then are used further by the same developer or licensed to others, e.g., Unreal Engine[6] or CryEngine[7]. Today, we also see game engines such as Unity3D[8], which are purely developed for licensing.

In the book 'Game Engine Architecture', Jason Gregory [65] gives a detailed description of the different modules included in many game engines and explains how they interact with each other and the system the game is developed for. Figure 2.6 shows a simplified version of the architecture described by Gregory [65, p. 39] where related modules are color coded. From the bottom up, we have the target system (grey) describing the actual hardware (e.g., PC, game consoles, or mobile devices), the operating system, and drivers. On top, we have core modules (green)

[2]`https://www.statista.com`
[3]`https://www.statista.com/statistics/246888/value-of-the-global-video-game-market/`
[4]`https://www.statista.com/statistics/552623/number-games-released-steam/`
[5]`https://store.steampowered.com`
[6]`https://www.unrealengine.com/en-US/`
[7]`https://www.cryengine.com/`
[8]`https://unity.com/`

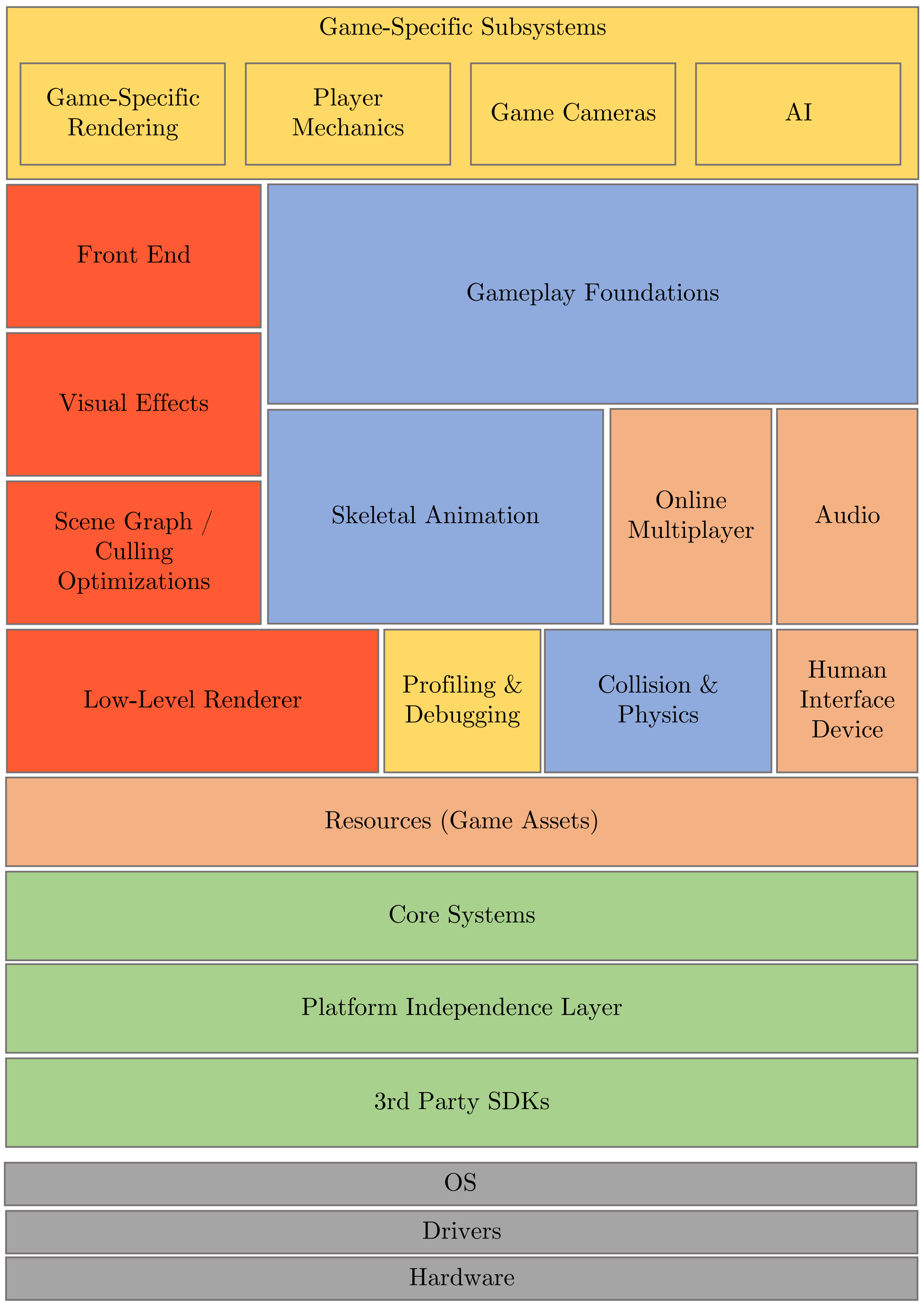


Figure 2.6.: Simplified depiction of a game engine architecture as described by Gregory [65]. Related elements of the architecture are grouped by colors in target system (grey), core modules (green), resource handling (orange), rendering (red), animation (blue), and developing tools and libraries (yellow).

including third party Software Development Kits (SDKs) (e.g., DirectX or PhysX), the platform independence layer (handling e.g., different file systems), and core libraries (e.g., for math functions, localization, or different parsers). Further, the game engine contains modules handling resources (orange) including game assets, input devices, audio, and network capabilities. The most computational heavy part of the game engine is the rendering (red). Gregory [65] divided this into the low-level render, visual effects, the scenes, and front end. Additionally, the game engine is responsible for animating objects (blue) as well as avatars and simulating the virtual environment and physics. Lastly, the engine offers debugging tools and gives developers a way to introduce own scripts and simulations (yellow) for e.g., artificial intelligence or game mechanics.

This breakdown of a game engine architecture shows the complexity, but also the opportunities provided to game developers. A game engine gives developers the possibility to increase their efficiency and reuse large parts of their code. In the same time they remain flexible, as they are not obligated to use all offered functionalities. In addition, many providers offer special licenses, which give developers access to the source code and allow them to modify it for their needs[9].

[9]`https://github.com/Unity-Technologies/UnityCsReference`

3. Related Work

The following chapter presents related work of this thesis. For the conducted literature review we discuss configurable pervasive middlewares and remote participation environments separately. In Section 3.1, we will first present the classification used throughout the review process. Afterwards, Section 3.2 discusses different middleware and consumer approaches allowing for a varying degree of human interference in the pervasive system. Following, in Section 3.3, different ways of remote participation are compared. Lastly, in Section 3.4, we will discuss the results of the literature review and where we identified the research gap.

3.1. Classifications

As the properties for pervasive middlewares as well as for the remote access and participation in smart environments differ greatly, we decided to introduce two independent classifications. Following, we will first take a closer look at our classification for configurable pervasive middlewares, before discussing the classification for remote participation environments. The focus is on the comparison of different configurable pervasive middlewares, as the remote participation is introduced as an extension to such systems and still relies on the functionalities offered by the middleware.

Figure 3.1 shows an overview of the classification used for the literature review on configurable pervasive middlewares. In total, five different classes are introduced: *user*,*scope*, *time*, *technique*, and *system properties*. The focus is on who is able to configure which part of the system at what point during runtime and how this configuration is carried out. Therefore, we first take a look at the *user* who is allowed to influence the behavior of the system. This can vary from the developer, who has to predefine configurations, to the user. In the latter case, we differentiate between domain experts who have a high domain knowledge or are specially trained, and end users without deeper knowledge. Regarding the *scope* of the

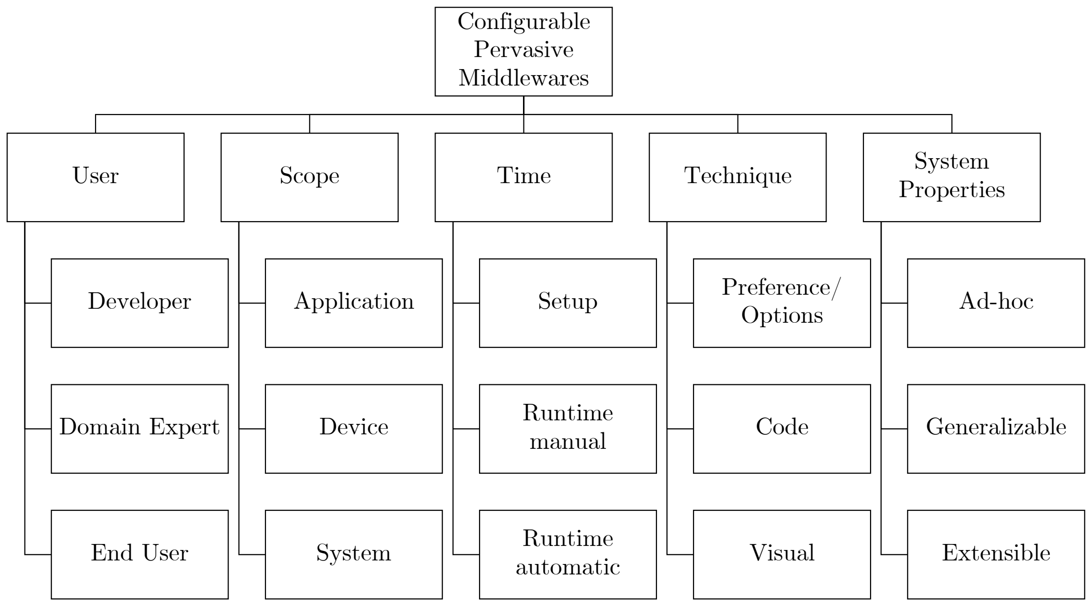


Figure 3.1.: Overview of the classification for configurable pervasive middlewares.

configuration, the complexity ranges from enabling the user to only influence the application he is using, the complete device, or several devices in the pervasive system. The third class describes the point in *time* when the configuration of the system can appear. Many systems need to be configured during the setup and are afterwards either fixed or changes in the system are handled completely automatically without any influence. We also classify systems as only configurable during setup even if it is technically not necessary to halt the system completely, in the case that the reconfiguration cannot be done without a large effort and thus is influencing the working experience or performance severely. This could be the case if the user needs to temporarily pause the system or if the manual process of configuring the system is very complex and requires a considerable amount of time. If the system allows for a configuration during runtime, we additionally differ between reconfiguration done completely manually by the user or automatically by the system. In the latter case, it is still important for us that the user is able to influence the behavior, e.g., by providing presets or preferences, which the system then uses for its automatism. Further, the *technique* used for configuring the system is considered. In the simplest case, the user has to provide preferences, access a simple options menu, or is prompted by the system with a simple choice if a context change is detected. In more complex systems, the user is required to write the configuration either as code during the development or in the form of scripts (e.g., in JavaScript Object Notation (JSON) or Extensible Markup Language (XML)). In the last case some systems support visual scripting for the configuration.

Lastly, the classification also takes some *properties* of the system into account. This includes whether it is possible to run the system ad-hoc without any infrastructure like permanent network or internet connectivity, specialized devices, or middleware services deployed on infrastructure devices. Further, the classification takes into account whether a system is generalizable and thus usable in different use cases and whether it is extensible with additional services and capabilities.

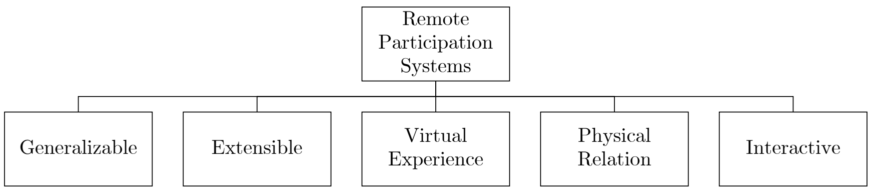


Figure 3.2.: Overview of the classification for remote participation systems.

Regarding remote participation systems, we focused on the functionalities they are able to offer to the user and on the overall experience. Figure 3.2 shows the classification used for those systems including the five used classes. Similar to the configurable pervasive middlewares, we discuss the *generalizablility* and *extensibility* of the systems. They should be able to adapt to a high amount of different use cases, e.g., remote meetings or participating in a lecture while abroad. Further, the classification takes into account whether the systems offer a similar experience to being physically present. Therefore, we first consider whether the system offers a *virtual and immersive experience* and second if it *relates to a physical environment* and connects devices in the physical and virtual world. Lastly, we discuss whether the system offers a degree of *interaction* to the user exceeding simple communication with other participants, e.g., by working on documents together or sharing files.

3.2. Configurable Pervasive Middlewares

Having introduced our classifications for the literature review, we now present the relevant approaches for configurable pervasive middlewares. While there is a plethora of context sensitive middlewares which automatically react, either reactively or proactively, to changes in the system itself or in its environment, we did not include them in the literature review. In general, the focus of the review was on middlewares that allow some degree of configuration and interference by the user. For further information on pervasive and context aware systems, we would like to refer to broader surveys [13, 84, 159]. While the conducted review spans almost two decades, it only discusses a subset of available approaches and does not aim to be exhaustive. The scope of introduced middleware was chosen to be as heterogeneous as possible, ranging from research based prototype systems to current smart home platforms advertised for end users. The following section is structured according to the classification shown in Figure 3.1 and each class is discussed in detail.

The degree of knowledge needed by the user to being able to configure the pervasive system varies widely between the different middlewares. In some cases, only the developers of applications and services are able to introduce configurations. Some middlewares (e.g., *MobiPADS* [29]) allow developers to write configurations as XML scripts, which are then automatically applied by the middleware in runtime. Other approaches like *GREEN* [177], *MUSIC* [165], *Dynamix* [27], or *EasyMeeting* [34] enable developers to program plugins or different components. These can then either be selected during setup (*Dynamix*), applied automatically during runtime (*GREEN* or *MUSIC*), or users are able to chose a predefined configuration manually (*EasyMeeting*). More modern commercial solutions like *Azure IoT Hub* [120] or *AWS IoT* [7] are more designed as interoperability systems. In these cases, developers are able to use middlewares like Node.js or Message Queuing Telemetry Transport (MQTT) for their applications and services. Despite using different middleware solutions the developers are able to connect the applications and services up to *Azure IoT Hub* or *AWS IoT* and configure their interaction with each other. Looking at middlewares which can be configured after the development phase, we have to differentiate between approaches which need a high amount of domain or IT knowledge and systems targeting actual end users.

In the first case, some middlewares like *dynamicTAO* [102] or *iROS* [97] require detailed understanding of the functioning of the middleware itself, which is hard to achieve for unfamiliar users without an IT background. *PCOM* [15, 71, 196] offers a visual way of configuring the applications. But as the structure and the terminology is very close to programming, it is difficult to grasp for untrained users. *Node-RED* [98] and *openHAB* [146] are two open source solutions published in recent years, which may allow for an easy configuration, but are complex to initially setup and thus also not usable for everyone. Further, systems like *Labscape* [66,67] and the healthcare system *ERMHAN* [150] offer an easy user interface but are targeted to physicians and laboratory employees. Systems designed for end users without further knowledge or experience with pervasive systems need to make some compromises regarding the configurability. Middlewares like *GUIDE* [36, 40] and the *Speakeasy Browser* [47, 136] are limited regarding the complexity of the configuration a user is able to create. With *GUIDE*, a tour planner, users are only able to provide the application with some preferences regarding tourist locations The tour planner then creates a tour accordingly. In the case of the Speakeasy Browser users can choose between predefined templates on how the system can be configured. Current end user systems like *IFTTT* [90] and *Apple HomeKit* [8], in contrast, limit the devices and services a user is able to utilize.

While most approaches allow the user to configure the complete pervasive system, some middlewares target only the application currently in use. *ProMWS* [30] and the system presented by *Hong et al.* [83] for instance allow the user to define preferences for services the application should utilize and then, during runtime, the middleware automatically predicts the preferred combination of services for the current context. *MundoCore* [5] enables applications to exchange middleware components by either loading them at startup or exchanging them during runtime to react to context changes, e.g., exchange the network component when switching from an ethernet to a bluetooth connection. Developers or administrators are able to influence which components should be used in which context via XML configuration files. Only two of the reviewd approaches focus on the configuration of devices instead of configuring applications only or whole systems. *ERMHAN* [150] for instance introduces a system to enable configurable services for health monitoring devices. *Dynamix* [27] is a framework for Android devices to enable the utilization of the different sensors in a smartphone to recognize the context of the user. Applications are able to request context information from the framework and users can influence which sensors should be enabled and with whom the information should be shared.

Regarding the point of time, many approaches require at least some configuration upon setup and before the system is actually in use. These are split into three different categories. First, some approaches like the Gaia middleware [162] based *GPM* [79] allow the configuration of the system only on setup. In this case, an administrator can create configuration files for the system either via scripting them by hand or with the help of a graphical tool and deploy them with the system. This is similar for interoperability focused systems like *Azure IoT Hub* [120] or *AWS IoT* [7], which require the developer to configure the interactions in the system during development or setup. Second, middlewares like *GREEN* [177] or *MundoCore* [5] enable developers or administrators to provide possible configurations during or before setup and then decide automatically on which of the configurations to apply based on context changes. Third, with *EasyMeeting* [34] and *Apple HomeKit* [8] we looked at two solutions which combine a configuration during setup with smaller manual reconfigurations during runtime. In both cases the majority of the setup is done beforehand, e.g., connecting different smart home devices to enable information exchange or assigning specific rights to devices and users. During runtime, the user then typically only applies smaller changes to the system such as turning on the lights in a room or changing what to show on a displaying device. Systems that allow for a reconfiguration during runtime are split into manual and automatic configuration in our classification. In the case of an automatic reconfiguration many systems, as with *GREEN* [177] and *MundoCore* [5] discussed, rely on several configuration created during setup which are then chosen automatically by the system. Further, *ProMWS* [30] and the approach of *Hong et al.* [83] reconfigure the system automatically based on the history of preferences the user provided in the past. Approaches relying completely on a manual reconfiguration during runtime often allow for only little influence by the user and are almost autonomous. *CARISMA* [26] offers users simple options menus where they are able to choose their preferences and change the thresholds for context changes in the system.

The next category of the classification shown in Figure 3.1 differentiates between the different possibilities and user interfaces provided by the systems for configuration. Most of the reviewed approaches utilize some kind of preference or options menu, which may enable an easy configuration but often does not provide a high complexity. The *Speakeasy Browser* [47, 136] for instance offers a web based menu system where users can choose predefined templates. The approach of *Byun et al.* [25] is even more simplistic. Here the system senses changes in the context and prompts the user with popup windows providing a simple explanation and a suggestion on how to react to the context change. The user in this case only has the option to answer with yes or no, but has no further influence on how exactly the system should adapt. Many other devices use code based configuration, either by programming and providing specialized plugins and components or by using script based configuration files with e.g., XML or JSON. As end users cannot be asked to learn programming, just to being able to configure the system, none of the reviewed code based approaches target end users. Each of the systems is either supposed to be configured by the developer or an administrator. *SOCAM* [68] for instance, is a service oriented middleware for the development of context aware services. In this case the developer of the service is able to define sets of rules with actions that are triggered by some context change in a configuration file using a proprietary language. The *RUNES* programming platform developed by Costa *et al.* [37] follows a similar approach. RUNES is a component based middleware where the developer is able to define rules on how the components should be exchanged automatically during runtime and how they are able to interact. With *PCOM* [15, 71, 196] and *Node-RED* [98] only two approaches are utilizing visual scripting for the configuration. Nonetheless, both are quite complex in their terminology and in their structure very similar to traditional programming. *Node-RED* is additionally hard to setup and integrate into a system without an IT background. Thus, even though they offer a very flexible and powerful way of configuring the system, they are not usable for end users without special knowledge.

Lastly, we reviewed some key properties of the middlewares which, depending on the use case, can influence the performance and usability of the system. Nearly all systems can be extended with further services and applications with only three notable exceptions. *GUIDE* [36, 40] is only designed as a tourist guide deployed on a Personal Digital Assistant (PDA) and thus is limited to the included services. The two modern consumer approaches *IFTTT* [90] and

Apple HomeKit [8] may offer a high number of functionalities and supported services and devices, but the user is limited to what the vendor is offering and is not able to extend the system on his own. All three approaches are also limited to their specialized use case as a tourist guide (*GUIDE*), for automation of web services (*IFTTT*), or for smart home systems (*Apple HomeKit*) and are not generalizable to other scenarios. Additionally, *EasyMeeting* [34] is limited to pervasive meeting rooms, *Labscape* [66, 67] is designed for laboratories, and *ERMHAN* [150] targets the healthcare sector. Lastly, with *PCOM* [15, 71, 196], *GREEN* [177], and *MundoCore* [5] only three of the systems can be seen as fully ad-hoc. All other approaches either rely on infrastructure devices or access to web based services. In this literature review we are only considering middlewares as ad-hoc, if they are able to form a spontaneous system between any peers without further input. This excludes approaches which offer for instance ad-hoc connection of user devices to the system, but are still relying on some infrastructure. An example for this is *Labscape* [66, 67], which allows the device of a researcher in a biology laboratory to connect automatically and transparently to the system. Nonetheless, the system requires some additional infrastructure for the device to connect to and needs connected laboratory equipment in order to support the use case. Further, systems like *IFTTT* [90], *Azure IoT Hub* [120], and *AWS IoT* [7] are cloud based and even if they offer some offline functionalities, they need a connection to the cloud in a regular interval, especially for reconfiguration. Other approaches like *Speakeasy Browser* [47, 136], *Dynamix* [27], or *openHAB* [146] deploy some of their middleware services on infrastructure devices. Other systems rely on the environment they are designed for, like the *GUIDE* [36, 40] system is deployed on PDAs and *EasyMeeting* [34] utilizes the devices present in a pervasive meeting room.

Table 3.1 summarizes our literature review on configurable pervasive middlewares. Only systems which allow at least some kind of interference by the user are included in this overview. Systems which are reconfiguring themselves purely automatic are not taken into account. Overall, the literature review shows a very heterogeneous group of approaches out of the last two decades.

	User			Scope			Time			Technique			Properties		
Project	**Developer**	**Domain Expert**	**End User**	**Application**	**Device**	**System**	**Setup**	**Runtime manual**	**Runtime automatic**	**Preference/Options**	**Code**	**Visual Scripting**	**Ad-hoc**	**Generalizable**	**Extensible**
GUIDE [36,40]			●	●			●			●					
dynamicTAO (CORBA) [102]	●	●		●				●		●				●	●
Speakeasy Browser [47,136]			●			●		●		●				●	●
GPM (Gaia) [79]		●				●	●			●	●				●
iROS [97]			●			●		●		●				●	●
MobiPADS [29]	●	●				●	●		●		●			●	●
Byun *et al.* [25]			●			●		●	●	●				●	●
CARISMA [26]			●	●				●		●				●	●
EasyMeeting [34]	●	●				●	●	●			●				●
PCOM [15,71,196]		●		●				●	●	●		●	●	●	●
Labscape [66,67]		●				●	●			●					●
SOCAM [68]	●					●	●				●			●	●
RUNES [37]	●					●			●		●			●	●
GREEN [177]	●	●				●	●		●		●		●	●	●
MundoCore [5]	●	●		●			●		●		●		●	●	●
Hong *et al.* [83]		●		●					●		●			●	●
MUSIC [165]	●			●			●		●		●			●	●
ProMWS [30]			●	●					●	●				●	●
ERMHAN [150]		●			●			●		●					●
Dynamix [27]	●		●		●		●			●				●	●
Node-RED [98]		●				●	●					●		●	●
Apple HomeKit [8]			●			●	●	●		●					
IFTTT [90]			●			●	●			●					
Azure IoT Hub [120]	●					●	●				●			●	●
AWS IoT [7]	●					●	●				●			●	●
openHAB [146]		●				●	●			●	●			●	●

Table 3.1.: Overview of existing approaches which allow the interference of users in the behavior of pervasive systems. None of the reviewed systems allow end users to reconfigure the complete pervasive system manually by offering visual scripting.

3.3. Remote Participation Systems

The discussed pervasive middlewares enable the construction of a physical smart environment and offer the users a varying degree of configurability depending on the used middleware. While this allows the users in e.g., a smart classroom or meeting room to utilize the system to increase usability and productivity, it is only an advantage for users who are physically present. Following, we will take a closer look at different approaches that allow users to participate remotely in all kinds of different events. While there is a multitude of different systems that allow people to communicate and share content remotely, we focused on a possible heterogeneous selection of approaches. The selected systems are analyzed using the classification introduced in Figure 3.2 and are sorted into the three categories of experience, communication, and collaboration systems. While experience systems allow users to only consume the provided content, communication systems go one step further. Here it is also possible to interact with other users and communicate with them either with pure speech, video calls, or gestures. Collaboration systems additionally allow the users to interact with different content in the system and to see the interaction of other users with the same content, e.g., editing a text file together.

The basic principle of experience systems, which is to enable people to consume content remotely is very old and has in principle already been introduced with television and radio many decades ago. But we argue, that the objective of these old systems is rather reporting remote physical events than recreating the actual experience of being there. This recreation has only emerged in the last years with the introduction of Virtual Reality (VR) devices like Oculus Rift [51] or HTC Vive [85] to the mass market. With *NextVR* [137] and *Oculus Venues* [52] we included two current approaches in the literature review. Both are very similar and only differentiate themselves in minor details. The main difference is that *NextVR* supports multiple platforms, while *Oculus Venues* is only usable on Oculus devices. Additionally, *NextVR* is even acts as a content provider for *Oculus Venues*. The goal of both approaches is to bring the experience of large public events to remote users by utilizing VR. Therefore, they deploy 360 degree cameras at concerts, sport events, or entertainment shows. Users at home are then able to view these events with their VR devices, either live or in some cases also at a later point in time as a recording. While, the immersion is much higher than classical television and it is possible to view the events from unusual angles, e.g., directly on the stage or behind the goal, it comes with the disadvantage that users are not able to move or interact within the virtual environment. Thus, it is only possible to jump between predefined camera positions and only rotational movement of the head is supported.

The next category contains systems that allow multiple users to communicate with each other. While traditional communication systems like telephone or the more modern Voice over IP (VoIP) systems like Skype[1] have been around for a long time, we are focusing on approaches that introduce further functionalities. All systems in this category provide some kind of virtual experience to the user. *I3DVC* [101] is a video conferencing system where all participating users sit in front of a desk with a life size screen at the other end of the table and participate in a virtual meeting. It is the only approach in this category that does not rely on three dimensional (3D) rendered virtual environments. Instead, it captures the user and the surrounding environment with 3D cameras and augments the captured video stream with virtual objects, e.g., a clock or a virtual desk. On top of communicating with the other participants, the system does not offer any additional interactivity and is also limited to the use case of video conferencing. *MASSIVE* [64] and *Virtual Society* [109] are two early attempts on virtual environments for collaborative working. Both render a 3D environment where the user is able to join with an avatar. The systems allow for interaction with other users either through communication via voice or text chat, or by interacting with virtual objects like white boards or projectors. While *MASSIVE* is limited to virtual meetings, it is possible to extend the *Virtual Society* system for different use cases. In their paper Lea *et al.* [109] describe how their system could be used e.g., for virtual meetings, games, or even shops. Both approaches are purely virtual and do not allow for the integration of additional devices or physical spaces. *Meetyoo* [119] and *Easy Virtual Fair* [46] are commercial platforms targeted at companies or public facilities. They offer the possibility to host large scale conferences, fairs, or exhibitions in a virtual environment. The organizer of the event can set up virtual booths showing different multimedia content and staff them with employees. Visitors of the virtual events are then able to walk through these booths with

[1]`https://www.skype.com/en/`

their avatars in the 3D environment and consume the content or chat with the staff. The interaction in this case is limited to communication with others and consuming the multimedia content. While it is possible for organizers to host a virtual event parallel to a real world event, it does not provide the same experience by directly mirroring it to the virtual environment. Both approaches are limited to this use case and only *Easy Virtual Fair* [46] is extensible with plugins in Hypertext Markup Language (HTML) 5.

The last category included in the literature review discusses systems that enable users to work cooperatively. While focusing on different use cases, all of the included approaches provide some kind of user interaction which exceeds pure communication. *ILIAS* [91] is an example for one of many (e.g., Moodle[2], Blackboard[3], or Canvas[4]) available systems offering educational institutions the possibility to communicate and share content with their students. *ILIAS* additionally offers modules like forums, quizzes, surveys, and wikis for students to enable collaboration and enhance the learning process. Nonetheless, the file sharing, communication, and collaboration functionalities are typically decoupled from the lecture and thus do not provide the same experience as attending the lecture in person. While *ILIAS* is limited in its use cases, it has been developed open source and therefore enables extensibility and allows for instance universities to develop their own modules to accommodate their teaching style and workflow. *Microsoft Teams* [121] and *Adobe Connect* [3] are collaborative working tools for the corporate environment. Both solutions rely on traditional communication (e.g., via webcams) and do not offer a virtual environment in which users are able to move. *Adobe Connect* targets online meetings and seminars. It therefore offers text, voice, and video chat and allows participants to collaboratively work on documents and share them. Additionally, *Adobe Connect* offers an Application Programming Interface (API) for developers and even ready to use applications to extend the offered software. While the focus of *Adobe Connect* is more on the communication between participants, *Microsoft Teams* focuses more on the collaborative working. Therefore, it offers direct integration of Microsoft's own Office Suite[5] where users are able to work on text, presentations, or spreadsheets.

[2]`https://moodle.com`
[3]`https://www.blackboard.com/index.html?nog=1&cc=US`
[4]`https://www.instructure.com/canvas/`
[5]`https://products.office.com/en-us/home?omkt=en-US&rtc=1`

Additionally, it incorporates a text chat for communication and Microsoft Skype[6] for voice and video chat. While it is possible to integrate Microsoft's own devices such as the Surface Hub[7], users are limited to devices officially supported by Microsoft. Further, the review includes *DOLPHIN* [181], *ActiveTheatre* [74], and *C-Slate* [94] as research based collaborative working systems. *DOLPHIN* was developed in 1994 to assist during meetings, It offers voice and video transmission as well as digital white boards. The main focus is to connect two groups in different meeting rooms and allow them to collaboratively work on the same white board. While similar systems are fairly common today, *DOLPHIN* offers an extensible architecture which allows developers to add further applications, something that even many modern systems like *Microsoft Teams* [121] do not offer. *C-Slate* [94] aims at cooperation between two distant users via a multi-touch and object recognition system. It therefore scans the working surface and tracks how the user is interacting with it. The interaction could be the movement on a board game or drawings on a paper. On the screen of the other user an augmented reality approach is used to overlay these interactions. *ActiveTheatre* [74] is designed to support surgeons in the operating room. A camera creates images and videos during the surgery and the surgeon is able to annotate them with text or speech and share them with others. Additionally, it is possible to review content while performing the surgery and thus get consultation from remote physicians. The system is extensible with further modules and the authors claim, that it is not limited to the use case of surgeries, but the introduced interaction techniques could also be useful in other collaborative working environments. Lastly, the only reviewed approach supporting a virtual experience is the virtual campus by De Lucia *et al.* [114] based on *Second Life* [112]. *Second Life* is a publicly available online virtual world developed by Linden Lab which allows people around the world to meet, socialize, build, and even trade in a 3D virtual environment. Further, it allows people to design and add virtual objects and buildings. De Lucia *et al.* used *Second Life* as a platform for their virtual campus with classrooms, a theater, and recreational areas with games like chess. The environments allow for presentations and meetings with the goal of collaborative learning. But this environment is not related to any physical classrooms or presentations.

[6] `https://www.skype.com/en/`
[7] `https://www.microsoft.com/en-us/surface/business/surface-hub-2`

Table 3.2 shows a summary of all reviewed remote participation systems. In total we included 14 different approaches sorted by the categories of experience, communication, and collaboration systems. These approaches are very different regarding their intended use cases and the solutions they offer. Still, none of them is able to connect a physical and virtual environment for multiple users to work collaboratively.

Project	**Category**	**Generalizable**	**Extensible**	**Virtual Experience**	**Device Integration**	**Interactive**
NextVR [137]	Experience			●	●	
Oculus Venues [52]	Experience			●	●	
MASSIVE [64]	Communication			●		●
Virtual Society [109]	Communication	●	●	●		●
I3DVC [101]	Communication			●	●	
Easy Virtual Fair [46]	Communication		●	●		
meetyoo [119]	Communication			●		
DOLPHIN [181]	Collaboration		●		●	●
ActiveTheatre [74]	Collaboration	●	●		●	●
C-Slate [94]	Collaboration	●			●	●
Second Life [112, 114]	Collaboration	●	●	●		●
Adobe Connect [3]	Collaboration		●			●
Microsoft Teams [121]	Collaboration				●	●
ILIAS [91]	Collaboration		●			●

Table 3.2.: Remote participation systems sorted by the categories of experience, communication, and collaboration systems. None of the discussed approaches achieve to provide an interactive virtual experience while simultaneously mirroring a physical environment.

3.4. Summary

During the analysis of configurable pervasive middlewares, we could observe that many of the approaches are either focused on the end user or offer a high amount of configurability. Middlewares allowing for a reconfiguration of the complete pervasive systems, e.g., *GPM* [79] or *MobiPADS* [29], are usually targeted at users with a high domain or IT knowledge. Systems allowing end users without prior knowledge to perform configurations with visual tools are on the other hand limiting what the user can influence and often only allow to configure small parts or to choose predefined settings, e.g., with the *Speakeasy Browser* [47, 136] or *GUIDE* [36, 40]. We argue that, with the rising popularity of smart environments in homes and working places, it is getting increasingly important to enable all users to set up the systems and reconfigure them during runtime if needed. Therefore, even end users without a background in IT or specific knowledge about the middleware should be able to configure the system for their current situation in an easy and efficient manner.

In the case of remote participation systems, with *NextVR* [137], *Oculus Venues* [52], and *I3DVC* [101], only three approaches offer a combination of a virtual experience and a physical environment. While *I3DVC* at least allows for communication between users, none of the three approaches offer any kind of further interaction. Thus, it is not possible for users to work collaboratively. All other approaches lack the connection of a physical and virtual environment and are therefore not able to give remote users the same experience, e.g., joining a live lecture at the home university while abroad.

In conclusion, none of the reviewed pervasive middlewares offer end users the possibility to use visual scripting for configuring the complete pervasive system to their needs manually during runtime. Further, none of the remote participation systems were able to offer an interactive virtual environment to enable collaboration between users, no matter if physically on location or not. The proposed PerFlow system is designed to tackle both issues. The PerFlow Middleware and PerFlow Tool offer visual scripting to allow end users to reconfigure the pervasive system and in combination with the PerFlow Virtual Extension remote participants can experience the same applications in a virtual environment.

4. Requirement Analysis

The following chapter derives the requirements for a configurable pervasive middleware considering the research questions introduced in Section 1.2. Therefore, we first describe a scenario for such a middleware and identify the stakeholders for this system. We will take a closer look on the expectations for each of the possible user groups. Based on these insights we will derive the functional and non-functional requirements for our system. In the remainder of the thesis these requirements serve as a basis for the design and implementation of the PerFlow system with all of its extensions.

4.1. Scenario

Following, we will introduce a scenario to identify the stakeholders and their requirements for the PerFlow system. While the pervasive middleware can be applied in all kinds of different use cases, we will take a deeper look at a pervasive learning environment. This example scenario can be transferred to other typical use cases, like smart meeting rooms or smart homes. The lecture or classroom scenario presents challenges which are typical for pervasive systems, namely highly heterogeneous devices, volatile connectivity, and the seamless integration of devices into the learning experience [169].

Figure 4.1 shows all relevant stakeholders and the interaction between them and the pervasive system. The scenario is divided into two different phases. First, the development of the services for the pervasive learning environment and the deployment of the system in lecture halls or classrooms. Second, the actual lesson hold in the learning environment. During the first phase two different stakeholders are involved: application/service developers and system administrators. Both pose different challenges to the pervasive middleware. The middleware is used by the developer to create applications and services for a high variety of heterogeneous devices and to enable the communication between them. These services are needed

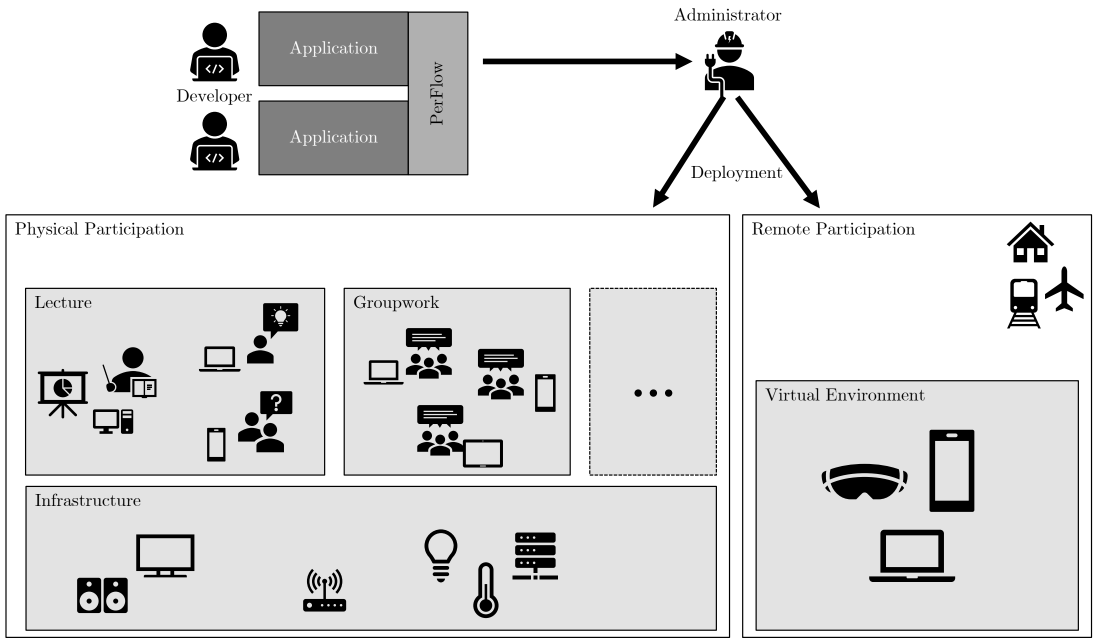


Figure 4.1.: Example scenario for a pervasive learning environment. Divided into the development and deployment, and the actual use during lectures.

to support the actual lecture and enhance the experience of the end users. The administrators in the next step are responsible for deploying the pervasive system in the actual learning environment by setting up the services and applications on the stationary devices and distributing them to the end users. Additionally, they define the behavior of the system during the actual lesson by providing initial configurations and defining the roles and the corresponding rights for the users.

During an actual lesson in such a pervasive lecture hall or classroom, we encounter two different user types. The lecturer gives the lesson and the students attending the lecture. The basic needs of both user groups are nearly identical. All users need the possibility to, for instance, share content like presentations or assignments, give or receive feedback regarding the lecture, or control infrastructure devices. The main difference between the two user groups is, that the lecturer should additionally have the possibility to influence the behavior of the system and control the students' access to different services. Further, the students can be physically on site or accessing the lesson remotely.

In total we are looking at five stakeholders: *developers*, *administrators*, *lecturers*, *local students*, and *remote students*. Further, we are dividing the pervasive learning environment into three device classes for *lecturer devices*, *student devices*, and *infrastructure devices*. While the device classes for the lecturer and students consist of nearly identical devices (mostly brought by the user) like notebooks, smartphones, and tablets, their purpose and the assigned tasks during a lecture may differ greatly. The *infrastructure devices* are highly heterogeneous and include displaying devices like TVs or projectors, servers, or lighting and shutters.

4.2. Functional Requirements

Based on the introduced scenario and the identified stakeholders, we are now deriving the requirements. These are split into functional ($\mathrm{R_F}$) and nonfunctional requirements ($\mathrm{R_{NF}}$). We start by introducing the functional requirements which, according to Laplante [108], describe the abilities of the system and which task it should be able to perform. In total we identified seven functional requirements that need to be supported for the aforementioned scenario.

$\mathbf{R_{F1}}$ - Information Exchange: One of the most important tasks of any pervasive middleware is to connect the different devices in the system and allow the communication between them. The PerFlow Middleware should enable developers to let their applications exchange information with each other. In the scenario the presentation application on the lecturers device should for instance be able to transmit presentations to the displays in the classroom.

$\mathbf{R_{F2}}$ - Runtime Reconfiguration: The main goal of the PerFlow Middleware is to configure the information flow between different devices and applications in a pervasive system. In contrast to many other solutions, the reconfiguration should affect not only single devices/applications but the complete system. Therefore, we require a syntax that allows to define rules for the information flow and determine who is able to send what kind of data to whom. These requirements should also not only be defined during the deployment phase, but the users should be able to influence them and reconfigure the pervasive system at runtime.

$\mathbf{R_{F3}}$ - Bundling Information Flow: To simplify the definition of rules for the end users it should be possible to bundle and organize the information flow between different devices and applications. Developers should get the possibility to define which API calls of their applications and services belong together and the users later only have to define the rules for these bundles and not for every single call. If a service for instance offers a control and a data connection the developer should define that these are both needed and the end user does not have to define rules for both.

R_{F4} - Heterogeneity Support: In our scenario, as in most pervasive systems, we are encountering very heterogeneous devices. This includes different device types e.g., smartphones, computers, or even smart lighting, as well as operating systems like Microsoft Windows, or Android. The middleware should be able to cope with this environment and connect all these devices. An additional challenge, and most important for our system, is the combination of static devices that offer their services continuously, as well as devices that might join and leave the system at any time. The PerFlow Middleware should not only enable the communication between all these systems. Users should also be able to use the PerFlow Tool for reconfiguring the system via visual scripting and the PerFlow Virtual Extension on a high variety of devices. The reconfiguration of the system should be possible on large touch devices like smart boards in lecture rooms, personal computers in meeting rooms, or tablets in an industry scenario. Further, the system is not only heterogeneous regarding the devices, but also with regard to the data communicated between the peers.

R_{F5} - Device Management: To allow runtime reconfiguration of the information flow we need an up-to-date list of the available devices and applications in the pervasive system. The middleware therefore has to handle device discovery and recognize changes in the system. It also needs to be able to address all devices and offer information about the system to the end user creating the rules for the pervasive system. This includes properties of the devices and applications (e.g., user groups of device types) and information about the data each application is able to receive or send.

R_{F6} - Access Control: While the main focus of the middleware is the reconfiguration of the pervasive system, not every end user should be allowed to influence the complete system. In our classroom scenario, the lecturer may want to reorganize the information flow in the system for group work between the students, but the students should not be able to hijack the projector during a lesson. Therefore, the administrators should be able to define during deployment who is allowed to reconfigure the system at runtime. Additionally, the access control should allow for the prioritization of users, so that, for instance, the lecturer is the only person allowed to reconfigure the system if present, but when he leaves the room the students are free to use the pervasive system as they want. Further, the middleware should ensure that the newest configuration is used in the complete system and no device is following an outdated set of rules. It is in the responsibility of the middleware to enforce the access control and keep track of the versioning for the configurations.

R_{F7} - Remote Access: Another goal of the system is to enable remote users a similar experience compared to the physically on site students. Therefore, the system should offer an virtual extension. Developers should be able to use the middleware to develop services and applications for a virtual lecture room. The middleware functionalities should be accessible from the virtual environment and allow the developers to mirror the functionalities of the services in the real lecture room. Further, the virtual middleware API should not be use case specific and as general as possible to enable the development of other virtual environments, like meeting rooms or museums.

4.3. Nonfunctional Requirements

In addition to the functional requirements, the system must also fulfill five non-functional requirements. While they do not specify functionalities offered by the system, they define properties and constraints of the implementation that effect the performance and usability [108].

R_{NF1} - Responsiveness: The middleware in our scenario is responsible to guarantee a seamless communication between devices. As the system is used during a live lecture, the information has to be shared nearly instantaneous in order to minimize the distraction of the lesson. Thus, it needs to control the information flow with a minimal overhead to avoid introducing a delay in the communication. Furthermore, the situations in pervasive systems such as smart lecture rooms might change any time and require spontaneous reconfiguration of the information flow. In our example scenario the access to a projector might initially only be granted to the lecturer device and during the lesson it might be reconfigured to be accessed by groups of students. The reconfiguration in this case must not introduce a noticeable delay and further may not affect the current exchange of data.

R_{NF2} - Fault Tolerance: The system should be fault tolerant in two different ways. First, devices may leave the pervasive system implicitly at any time without prior information. Therefore, the middleware should not depend on a central device that constitutes a single point of failure. Thus, the rules which define the information flow must be maintained even though the administrating lecturer has temporarily left the system. Additionally, the middleware should not lose any data. Second, while reconfiguring the middleware, the user should not be able to introduce faulty configurations. As the communication in the system is reliant on a correctly configured information flow, the middleware should hinder reconfigurations which cannot be interpreted due to syntactical errors or executed due to semantical errors.

R_{NF3} - Generalizability: While we are using the scenario of a pervasive lecture room as an example during this thesis, the middleware needs to allow the user to configure the information flow in all different kinds of scenarios. As pervasive systems are spreading more users are getting in touch with them in e.g., smart homes, airports, or offices. Unexperienced users still need the ability to influence the system and thus profit from PerFlow. Therefore, the basic functions of the middleware should not be use case specific and the developer should be able to customize the middleware to his needs.

R_{NF4} - Usability: We argue that the success of pervasive systems also depends on their usability. Even unskilled end users need to be able to configure the information flow in order to use the full potential of smart environments. If only trained system administrators can setup the information flow, the environment will become static and eventually less attractive. Thus, the ease of use is among the most critical requirements and calls for special attention in the design of a pervasive middleware. This also includes the introduced visual scripting tool and virtual environment.

$\mathbf{R_{NF5}}$ - Extensibility: While for many applications the communication of basic data types like integers or strings may be sufficient, especially today we are facing many different multimedia types. For instance, in our scenario the lecturer may want to show a video or distribute a presentation. Therefore, the middleware should allow the developer an easy extension with new data types to enable for instance the distribution of image, video, or audio data. The middleware should offer the developers the possibility to define their own serialization routines for their custom objects and therefore unlimited possibilities regarding the handled data types.

5. System Model

The following chapter will describe the system model of PerFlow.Thus, we will introduce the terminology used throughout this thesis and describe all parts constituting the pervasive system. While we are using a pervasive lecture room as the main example the system discussed in the following chapter is described in a broader term and without a specific scenario as a foundation.

In traditional pervasive systems or smart environments like Gaia [162] or Aura [180] the services are provided by the infrastructure. Additionally, servers or devices located in the infrastructure are also responsible for controlling the system. Thus, to be able to utilize the pervasive system, some kind of dedicated device has to be deployed beforehand to offer basic system services. In contrast, the proposed PerFlow system follows the concept of smart peer groups discussed in [72]. The middleware functionalities in PerFlow are completely decentralized and allow for the building of ad-hoc systems between the mobile peers provided by users. Nonetheless, it is also possible to incorporate nearby infrastructure devices and their offered services, but they are not crucial for a functioning system.

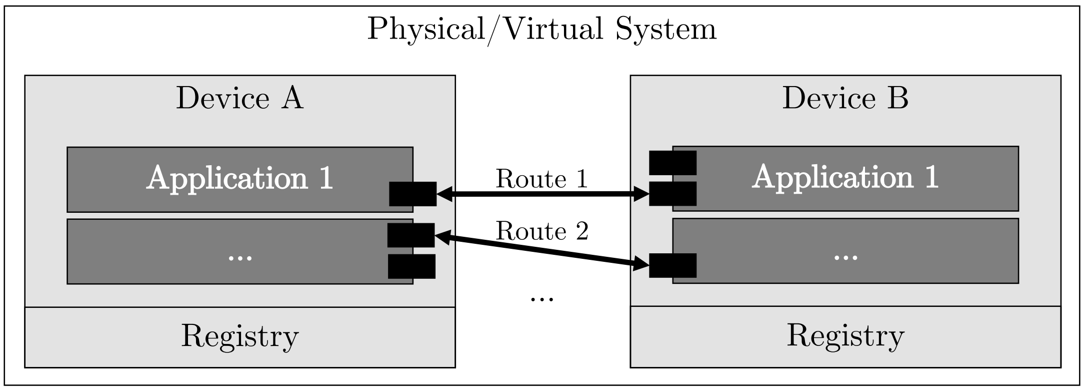


Figure 5.1.: The system model for PerFlow depicting two devices with multiple applications and one registry. Each application may have several connectors linked with routes for information exchange.

As depicted in Figure 5.1, our system consists of a *physical* and *virtual environment*. Both may contain multiple *devices* with each containing several *applications* with *connectors* for in- and outgoing communication. Additionally, each device administers a *registry* providing information about the system for developers and administrators. The *connectors* of two applications may be connected via a *route* which allows communication between them. While these routes describe the communication between single applications, *information flow* describes the exchange of data in a broader term e.g., the application of one user is able to send images to applications of all users regardless of the specific devices involved. The user is able to formulate this behavior in so called *rules*, which are then interpreted by the middleware to create the *routes*. To enable the communication between devices we rely on a *communication middleware*. The virtual environment is connected to this middleware via the *virtual proxy*. Following, we will take a closer look on each of these elements.

Device: The system may include many heterogeneous devices e.g., personal computational devices like smartphones or laptops, and infrastructure devices, such as projectors, servers, or smart lighting. These devices may also be brought by the user and thus their availability is not always given. Each device may join or leave the system at any moment.

Application: Any device may run several applications using the PerFlow Middleware to communicate with others. An application may send and/or receive data from or to an other application. The scope of functions may range from simple services, such as presentation services showing content on a screen or providing a temperature reading, to more complex applications combining different functionalities and communicating with multiple other devices simultaneously.

Connector: Each of the applications may have several connectors describing the data it is capable of sending or receiving. A connector always handles one single data type. It can be ingoing for receiving or outgoing for sending data in the system. Additionally, the connector may be pushing the data automatically into the system or wait until the data gets pulled from others. In a similar fashion an ingoing connector can automatically pull the needed data from others or wait for it.

Rule: Rules can be defined by the user and describe the information flow in the pervasive system in terms of specific properties of devices and applications. The rules are formulated on the system level and may influence complete groups of devices at once. The middleware is then responsible for the interpretation of these rules as specific routes between two devices.

Route: A route describes the connection between two applications. It allows the data exchange between an outgoing and ingoing connector. To achieve this, it is not relevant if the connectors are actively or passively sending/receiving data, the PerFlow Middleware is responsible for handling the data exchange accordingly. For a valid route it is only important that both connectors are specified for the same data type.

Information Flow: While routes describe the communication between two applications, the information flow describes the communication on a system level. The information flow describes the exchange of data between multiple devices and it is also possible to bundle several routes. As an example the information flow between a file server and notebooks belonging to end users may contain routes for control messages so that end users can request files and routes for the data transmission.

Registry: Each device in the system is managing a local registry. It stores and provides information about the capabilities of the device and the local applications, including all available connectors. These information can be accessed and extended by application developers and they can be used by administrators and users to reconfigure the information flow.

Physical Environment: This environment represents an actual physical room equipped with a possible large amount of heterogeneous devices. These devices may be user devices like notebooks or tablets, or infrastructure devices, such as smart lighting or TVs. The devices form a typical pervasive system with the goal to support the users in their daily tasks by offering services depending on the smart environment. This could be for instance in a pervasive meeting room or in a smart home.

Virtual Environment: The goal of the virtual environment is to provide the same experience to remote users as if they were physically available in the actual pervasive environment. Therefore, developers need access to the functionalities of the PerFlow Middleware within the virtual environment to be able to develop similar applications to the ones used in the physical environment. Further, for the development of the virtual environment we rely on existing frameworks for the development of 3D environments (so called game engines).

Virtual Proxy: To enable the access to the middleware functions from within the virtual environment we need a proxy. It mediates between the PerFlow Middleware and the applications developed in the game engine. The main goal of the proxy is to marshal and demarshal the data for the use in the virtual environment and to handover the function calls to the middleware. The proxy itself should not introduce any further logic or registries for information management.

Communication Middleware: As pervasive systems are already in development for many years, there are multitude of middlewares available.Within this thesis we are focusing on the development and implementation of a middleware for the realtime reconfiguration for such a system. Therefore, we are using an existing middleware for the device discovery and communication within the pervasive system. The PerFlow Middleware is developed independently and should allow for an exchange of the underlying communication middleware.

6. Middleware Design

Using the system model, we discussed the terminology and environment which we encounter with the PerFlow system in Chapter 5. Further, we introduced the scenario we are facing with our middleware in Section 4.1, before we derived the requirements for PerFlow in 4.2 and 4.3. In the following chapter, we now want to discuss how we are tackling these requirements and explain our approach for a runtime configurable pervasive middleware in detail. Additionally, the PerFlow Tool for user created configurations and the PerFlow Virtual Extension for remote users will be presented.

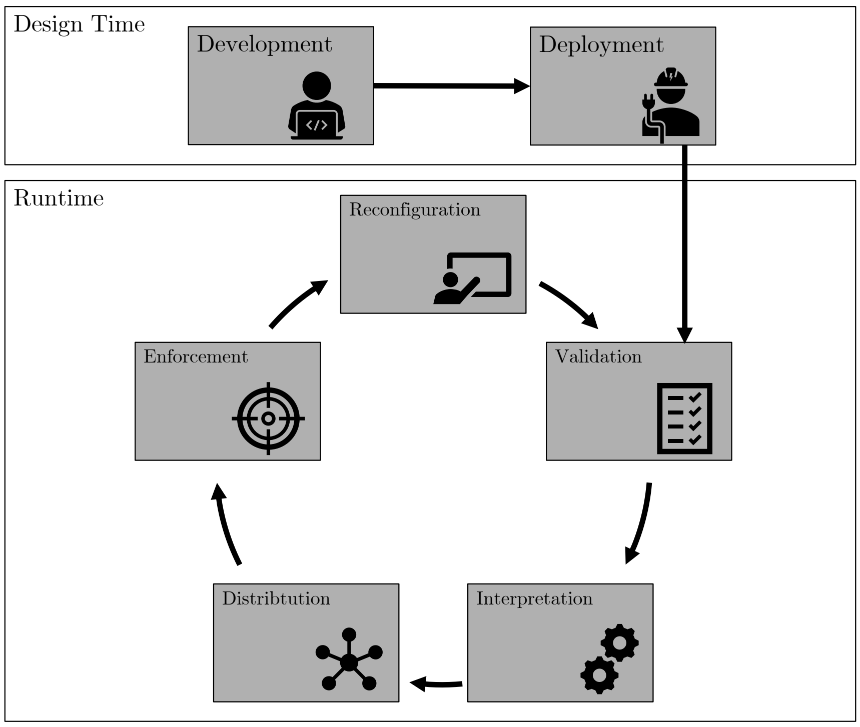


Figure 6.1.: The complete life cylce of the PerFlow system starting with the design time, including the development of applications and the deployment by an administrator. During runtime the system can be reconfigured by users.

In Figure 6.1 the complete process of the reconfiguration can be seen. At the beginning, the application developers have to specify their API and register it at startup with the middleware. If further functionalities get available or existing get obsolete during runtime, the registry entries can be altered. During deployment, the administrator should provide an initial configuration for the system to ensure, so that it is usable from the start on without requiring a beforehand setup by the users. Additionally, the administrator may also specify who is able to reconfigure the system during runtime. Further, if a new configuration is provided, the middleware is responsible for checking the validity and distributing it to all devices in the system. At runtime, the middleware has then to enforce the currently available configuration. To enable the end user to create new configurations on their own and without any advanced knowledge in the field of information technology or even pervasive systems, a visual scripting language was developed. This language can then be used to create rules in a visual scripting tool connected to the middleware. One additional requirement was the integration of remote users. Therefore, we expended the system with a virtual environment. The remainder of the chapter is dedicated in explaining the design of each of the aforementioned steps. Further, it is also structured according to the process explained above. The following chapter is based on and extends [132][1] and [131][2].

6.1. PerFlow Architecture

The following section will provide a complete overview of the architecture used for the PerFlow system before going into the details on the separate parts in the remainder of this chapter.

Figure 6.2 shows the overall system architecture for the PerFlow system. It is split into three major parts: PerFlow Middleware, PerFlow Tool, and PerFlow Virtual Extension. The system utilizes a communication middleware for the transmission of messages and device handling. Each physical device in the system has to run at least the PerFlow Middleware combined with the communication middleware. This module handles the information about the pervasive system and contains the main functionality needed to reconfigure

[1] [132] is joint work with M. Pfannemüller, J. Edinger, and C. Becker
[2] [131] is joint work with C. Krupitzer and C. Becker

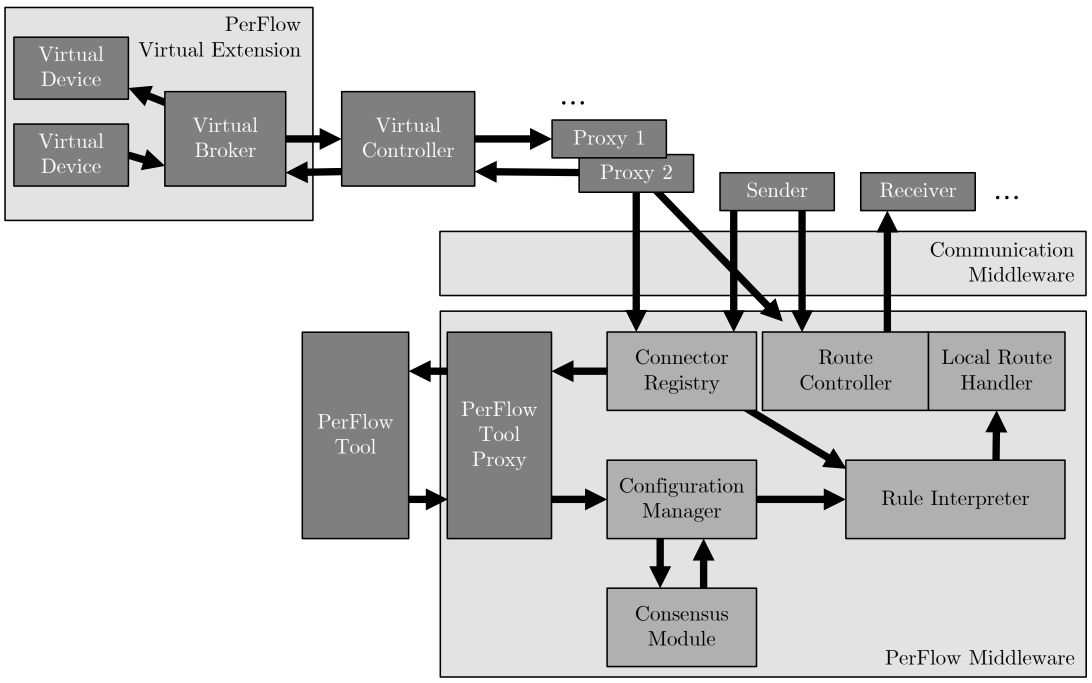


Figure 6.2.: The overall system architecture for PerFlow consisting of the three major modules for the PerFlow Middleware, PerFlow Tool, and PerFlow Virtual Extension. To enable the message exchange between all parts of the system a communication middleware is used.

the information flow in the middleware during runtime. The *Connector Registry* handles the current status of the pervasive device. Further, the *Configuration Manager* is used to store the current valid configuration and distribute new configurations either in a naive approach with low overhead or, by using the *Consensus Module*, in a more complex way supporting access control. The information stored in the *Connector Registry* and *Configuration Manager* is then used by the *Rule Interpreter*, which uses the combined information to generate communication routes for each device. These are then stored in the *Local Route Handler* of the related device and are used by the *Route Controller* to determine the target for each message and transmit the outgoing information accordingly via the communication middleware. Developers directly access the *Connector Registry* to publish which information their applications are able to receive or send. Additionally, they need to implement the corresponding interfaces offered by the *Route Controller* to distribute or listen for the data.

The PerFlow Tool and PerFlow Virtual Extension are optional and the users may start them if they need them. The proxy for the visual scripting tool connects to the *Connector Registry* and *Configuration Manager*. It forwards the information stored in the registry to the actual PerFlow Tool and receives and forwards the new configurations created by the user. With the PerFlow Virtual Extension it is possible for developers to implement applications inside a virtual environment. They are able to access the PerFlow Middleware in the same fashion as developers implementing for the physical pervasive system via the *Virtual Broker*. The *Virtual Broker* hands the API calls over to the *Virtual Controller*, which starts one proxy for every virtual device in the virtual environment. These proxies access the *Connector Registry* and *Route Controller* as any other application in the virtual environment.

6.2. Structuring the Information Flow

To enable the reconfiguration of the system the user creating the configuration and the middleware enforcing it need information about the available devices and applications. Responsible for collecting and managing this information is the *Connector Registry*. The goal is to enable users to influence the information flow between different applications by offering them the possibility to define witch applications can share witch information according to their needs. While there is some information which is always needed by the system to function properly, developers should also have the possibility to introduce their own properties. Thus, the system is not tailored to a specific use case, but developers can choose the needed information to support their scenario. Additionally, the registry is responsible for providing the collected information to other peers in the system. In the following sections we are discussing what information is needed about the system, how the registry handles the information locally, and how it is distributed to the rest of the system.

6.2.1. Structure

The *Connector Registry* is designed to support developers as well as administrators and users. These stakeholders have vastly different demands to the registry which are taken into account during the design. Regarding the handled information, the developers need to know which data they can expect and how it is serialized to be able to use it in their applications. Administrators and users on the other hand, require additional information about how the system is structured and what the function and role of each application in the system is. This enables them to make decisions on how the data in the system should be distributed and what devices or applications should be able to communicate with each other. Therefore, they for example want to know, which user group a device belongs to, which device type it is, or where it is located. As this information is strongly dependent on the use case, the design should be flexible regarding additional properties.

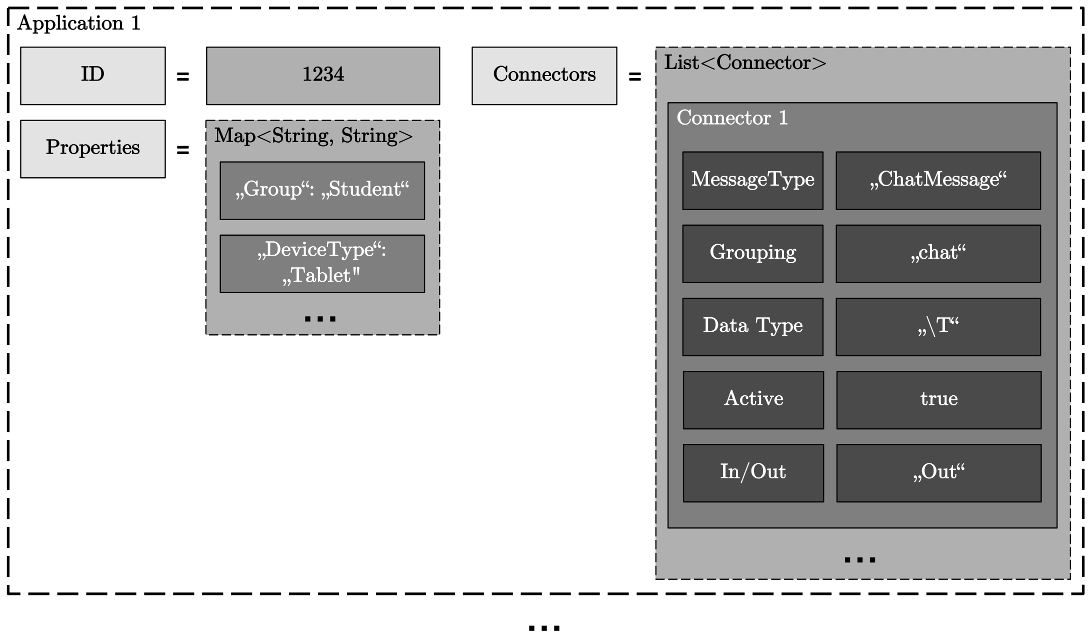


Figure 6.3.: The information saved by the local *Connector Registry* about one specific application containing its ID, properties and a list with all available connectors.

Figure 6.3 depicts the information stored in the *Connector Registry.* Here the local registry for one device is shown containing information about one application. For each device it is possible to store several applications. The local *Connector Registry* provides access to this information to the user of the device and developers are able to add new or change the stored information about their application. If a complete view on the whole pervasive system is needed, the information stored in the registries of each device can be combined. For each application the registry stores an *Identifier (ID)*, *Properties*, and *Connectors*. Following, we will discuss each of these values in more detail.

ID: As the used communication middleware is responsible for the device discovery and handling we are utilizing it also to address specific devices. To be able to work with multiple applications on each device in the system, we further introduce an ID for each application. These IDs are unique within the context of one device. With the combination of the device address provided by the communication middleware and the application ID introduced by PerFlow it is possible to address each individual application in the system.

Properties: The properties are used to describe the context of the application in detail. As the context may vary vastly with regard to the use case the system faces, these properties should not be predefined during the development of the middleware. Therefore, the PerFlow Middleware offers the possibility to specify custom key/value pairs. Thus, developers are able to add for instance "location" as a key and "meeting room" or "classroom" as the value. Properties can be used to identify groups of devices or applications e.g., with the key "role" and the value "student". The properties can be added to the *Connector Registry* by the developer of the applications or by the administrator during deployment. Further, they can also be altered during runtime, for instance to change the location as soon as the device moves. Additionally, the developer can give the end user access to specific properties, so that he is able to alter them during runtime, e.g., for setting their own user name. It is also possible to introduce a system for context awareness [171], which automatically sets specific properties based on a detected context change. A Global Positioning System (GPS) sensor could for instance be used to detect a change in location and update the according property of the application.

Connectors: The connectors for an application describe the possible incoming and outgoing information. Each application may have multiple connectors. The *Connector Registry* is used by the middleware to determine which connectors can be used to communicate via a route. Additionally, users and administrators need this information to be able to create a valid configuration for the system. The information about the connectors needs to be supplied by the developer and is registered at the registry during the start of the application or when additional functionalities with new connectors get available. Each connector is described with six attributes, as shown in Figure 6.3. *Data type*, *active*, and *in/out* are used by the middleware to determine how to create a route between two connectors and to serialize the data correctly. The data type therefore specifies, which primitive value or serializable object has to be expected for the transmission. Further, it is described, if the connector is sending or receiving data and if it actively does so or if the opposing connector is responsible to retrieve or push the data. *Message type* and *grouping* are mainly used to provide users and administrators information about the system and to support them during the configuration. The message type therefore describes what kind of information is communicated via the connector e.g., "chat message" or "presentation". The grouping attribute is optional and gives the developers the possibility to group several connectors with one common keyword. Thus, administrators and users can connect multiple connectors at once, if the developers decides that it would not make sense to use them separately. This could for instance be the case for presentations. Here one connector is used for controlling the slides and another one for transmitting the actual content of the presentation, but for both the *grouping* attribute is "presentation".

6.2.2. Local Connector Registry

After discussing the information needed by the PerFlow Middleware to offer runtime reconfiguration, we now want to concentrate on how these information are handled. In this section, we will first take a look on the *Connector Registry* itself, which is executed locally on each device in the system. Afterwards, we discuss how the registries exchange their information in order to offer a complete overview of the system to the administrator or user.

The *Connector Registry* has three main tasks in the proposed system: *collecting*, *providing*, and *distributing* information about a device and its applications. Therefore, it offers the four basic functions of the Create, Read, Update, and Delete (CRUD) principle for storing and handling data [118]. But as the context of the system may change with every startup, the information is not stored by the registry in a persistent manner and the application has to provide its current information while registering with the PerFlow Middleware. In Listing 6.1 the API of the *Connector Registry* is shown, divided into the CRUD functionalities.

```java
    // create/update
    public void registerApplication(ID id, Map<String, String>
        properties);
    public void registerConnector(ID id, boolean outgoing, boolean
        active, boolean allowsArray, String messageType, String
        description, String dataType);

    // delete
    public void removeApplication(ID id);
    public void removeConnector(ID id, String messageType, String
        dataType);

    // read
    public Application[] localApplications();
    public Application localApplication(ID id);
    public Application[] remoteApplication();

    // locking
    public void lockRegistry();
    public void unlockRegistry();
```

Listing 6.1: The interface of the Connector Registry showing the CRUD functionalities for handling the connectors of the different applications running on a device. The interface is defined in Java.

First, the API allows an application to create a new entry by registering itself and providing its ID and properties. After at least one application is registered, it is possible to also register connectors for it by providing the attributes discussed before. An application is always identified by its ID and a connector of a specific application can be identified by the combination of message and data type. If an already existing application or connector is registered, it is not stored as a duplicate but instead, the existing entry is updated. Further, the registry allows for deleting applications or connectors. If an application is deleted, the *Connector Registry* automatically deletes all connectors which may exist and are assigned to this application. Next, the registry offers the possibility to obtain the data of all registered applications and connectors. It is possible to retrieve the complete list of all registered objects or only the data for one specific application. Further, we distinguish between reading the data locally or polling the data from the complete pervasive system. Lastly, the registry can be locked for registering, updating, or deleting applications and connectors. This is necessary during the reconfiguration of the system. For example, in case of a application first reading all available connectors from a registry and another application then deleting a connector, a new route would could be created for a no longer existing connector. To prevent interferences during the reconfiguration, all registries in the system are locked at the beginning of the process. As long as the lock is active, all requested changes are buffered. After the lock is lifted, all of the stored requests are executed in bulk.

6.2.3. Information Exchange

Until now we are able to store and retrieve the information about the applications and connectors on the local device. In order to being able to reconfigure the complete pervasive system, the administrator or user needs a complete view on the system with information about all devices. Therefore, we need the possibility to read the data from all registries in the system.

To achieve this, the Connector Registry is designed as a service for the underlying communication middleware. The registries have a fixed application ID identical on all devices and are started automatically together with the PerFlow Middleware. Additionally, the communication middleware can be used to retrieve a list of all available devices in the pervasive system. Thus, with the device ID and application ID we are able to address each Connector Registry in the system directly. If a complete image of the system is requested at once of the registries, it first adds its own stored information to the result and then requests the data from all other registries via the communication middleware. These requests are not depending on each other and are therefore handled in parallel. After receiving a result or timeout from each registry, the complete data is aggregated and returned. As we always need a complete view on the system to reconfigure it, the registry does not offer the possibility to only request selected information from other devices.

6.3. Configuration of the Information Flow

After defining which information about the system is stored by the Connector Registry and provided by the PerFlow Middleware, we take a deeper look on how this enables us to configure the information flow in the system. In the following Section we explain how the information about the devices and applications is used to create configurations. Therefore, we first take a look on how a configuration is defined as a set of rules. Then, the structure of the routes used by each device to determine where to send its information during runtime is discussed. Afterwards, the interpretation of the rules to create the routes for each device is explained in detail. And lastly, we talk about the process of delivering the newly defined routes to each device in the pervasive system.

6.3.1. Rule Definition

To determine who can send information to whom, the middleware needs to configure the information flow in the pervasive system. Therefore, the PerFlow Middleware introduces one central configuration containing the rules for the complete system. Administrators should be able to create a standard configuration during the deployment phase, which is then loaded and applied during the start of the pervasive system. With the goal of supporting users without IT knowledge in creating new rules, we will introduce our visual scripting tool in Section 6.6. Further, the configuration should be as lightweight as possible due to the fact that it may change multiple times during the runtime of the system and the rules have to be distributed to all devices. Therefore, we chose JSON [99] over XML [193] as the data format for the rule definition, as it is easily human readable and more lightweight [142].

At any given time there is only one active configuration in the pervasive system which is stored on each device. A configuration may consist of multiple rules defining the information flow in the system. One rule has attributes defining the *sender* and *receiver*, the *transmitted information*, and three different *filters*. Following, we will take a closer look on these attributes.

Sender and Receiver: To define the start and end of the information flow we need to supply the middleware with the related devices as the sender and receiver. If only specific devices should be able to use this route, we can define them with a list of the combined device and application IDs. As the administrators or users may want to define multiple devices as sender or receiver based on their properties it is possible to specify "ALL" as the sender or receiver. In this case every device in the system is able to use the route, if no further limitations in the form of filters are provided.

Transmitted Information: Every rule also needs to specify which information should be handled. Therefore, three different attributes have to be provided. First, the message type describes which kind of information is influenced by this rule e.g., "chat message" or "presentation". Second, the attributes for data type and array tell which exact data is transmitted between sender and receiver. The main purpose of these attributes is for the PerFlow Middleware to know how to serialize and deserialize the data. Further, this information may be useful for the administrator or user, if several connectors are bundled under one message type. For instance in the case of presentations, we may have a connector for images and one for controlling the slides. In such a case it is possible to choose a specific connector via its data type or leave the value empty to include all bundled connectors.

Filter: In total, each rule may have up to three different filters. The first two influence the possible senders and receivers. These filters contain a list of properties a device needs to possess in order to act as a sender or receiver. If multiple properties are provided in a filter, these are evaluated with a logical "and". In the smart classroom scenario this would, for instance, allow all devices with the properties "group" equal to "lecturer" and "device type" equal to "PC" to send handouts. If the properties should be connected with a logical "or", two separate rules have to be defined. Further, the PerFlow Middleware supports context filters. These are not used during the interpretation of the rules to generate the routes but instead, they are evaluated during runtime. Each context filter is applied to the transmitted data and allows to react to the context of the system in real time. For instance, could a context filter of ">20" be defined for the route between a sensor and a smart light bulb to only forward the information when the light level exceeds 20.

```
 1 {"Rules":
 2   [
 3     {
 4       "messageType": "chat message",
 5       "dataType": "\T",
 6       "array": false,
 7       "sender": ["ALL"],
 8       "senderFilters": {
 9         "group": "student"
10       },
11       "receiver": ["ALL"],
12       "receiverFilters": {
13         "group": "lecturer",
14         "deviceType": "pc"
15       },
16       "contextFilters": []
17     },
18     ...
19   ]
20 }
```

Listing 6.2: One example rule in the JSON format defining the content of the message, sender, and receiver with their filters. A configuration can contain several rules.

Listing 6.2 shows a single example rule in the JSON format. Line 3-5 show that the rule handles chat messages with strings as a data type ("\T") and does not allow the transmission of arrays. The sender is defined in the lines 6-9 and allows all devices of the group "student" to send the messages. Further, the rule defines in the lines 10-14, that the information will be send to all devices of the type "PC" in the group "lecturer". Lastly, context filters are not applied, as shown in line 15. Each configuration may contain a multitude of such rules. While these rules are describing the information in the complete pervasive system, they need to be interpreted for each device.

6.3.2. Route Definition

After the interpretation of the rule each device is updated with its specific routes. These routes describe, which information this device should send to which other device. A device may store a list with a multitude of routes. The *Local Route Handler* is responsible for managing and storing the routes. Further, the routes are neither relevant to the developer nor to the administrator or user. Their are only used internally by the PerFlow Middleware. The developers only call the middleware and handover their information. The middleware then decides how it should deliver the information based on the routes. This process is described in more detail in Section 6.5.

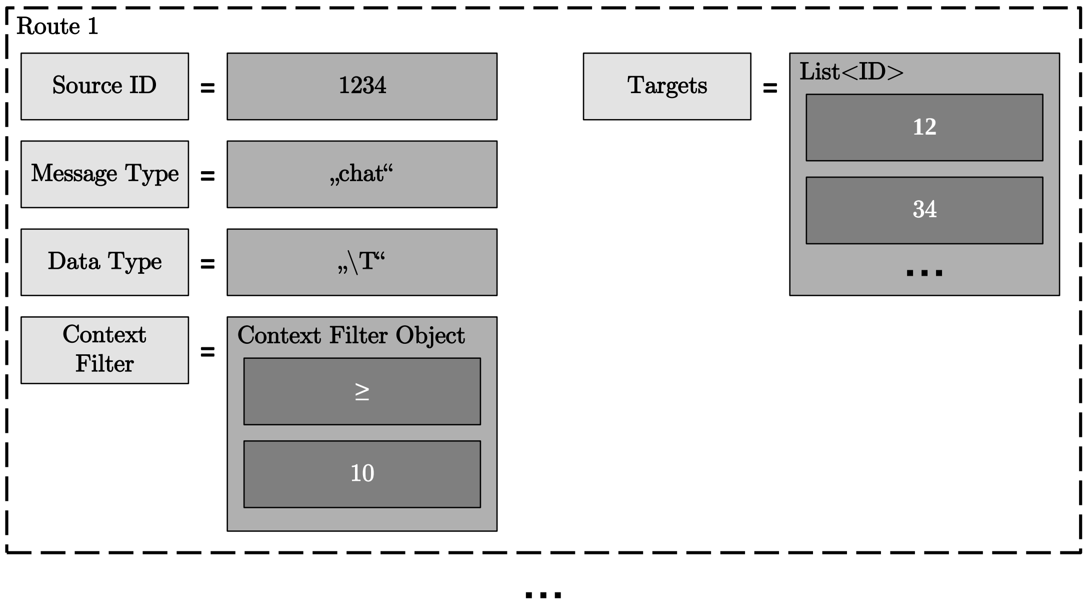


Figure 6.4.: Each route consists of five attributes defining which applications send the specific information to how many target applications and their IDs.

A route is defined by six attributes as shown in Figure 6.4. As a device may run several applications, the route first has to specify which of these is the source. Afterwards, the message type defines, which information is handled by this route and the data type is used by the middleware to choose the correct serialization method. If a rule uses the bundling mechanic to combine several connectors with different data types in one rule, this results in multiple separate routes after the interpretation. Next, the route may contain an optional list of context filters, each containing an operator and value. These are evaluated by the middleware as soon as information is transmitted via this route. The data is therefore compared with the value based on the operator deciding whether the middleware should forward the information to the targets. Lastly, the route contains a list with one or multiple targets. If this specific route is applicable to the transmitted information, it gets delivered to all the targets in the list.

6.3.3. Rule Interpretation

Now that PerFlow is aware of all the devices in the system and their capabilities and is provided with a set of rules on how the system should be configured, it is possible to update each device with its routes. Therefore, the rules need to be interpreted based on the current status of the system by the *Rule Interpreter*. Further, this interpretation needs to be executed whenever the configuration or the composition of the pervasive system changes. The device detecting the change is responsible to run the interpretation in such a case and distributes the new routes to the corresponding devices. Thus, only one device has the computational load. PerFlow also distinguishes between a reconfiguration with new rules and a change in the pervasive system e.g., a new device joining or an application closing and deregistering its connectors. In the first case, the PerFlow Middleware always carries out the interpretation for the complete system and updates all devices with their new set of routes. For the second case, the middleware only updates the devices affected by the small change. This is due to the fact, that for an incremental change in the case of a new rule set the interpreter would first need to check all devices and applications, if they are affected, which would be a similar effort to calculating all routes, even if some of the existing routes could be reused.

In Figure 6.5 the complete process for the interpretation of the rules is shown. To be able to interpret the rules for the configuration, the interpreter always needs a complete overview of the pervasive system. The Rule Interpreter retrieves the current system status from its local Connector Registry, which therefore polls the registries on all remote devices and generates a complete list of all applications with their connectors. We differ between two sets of device information, the input

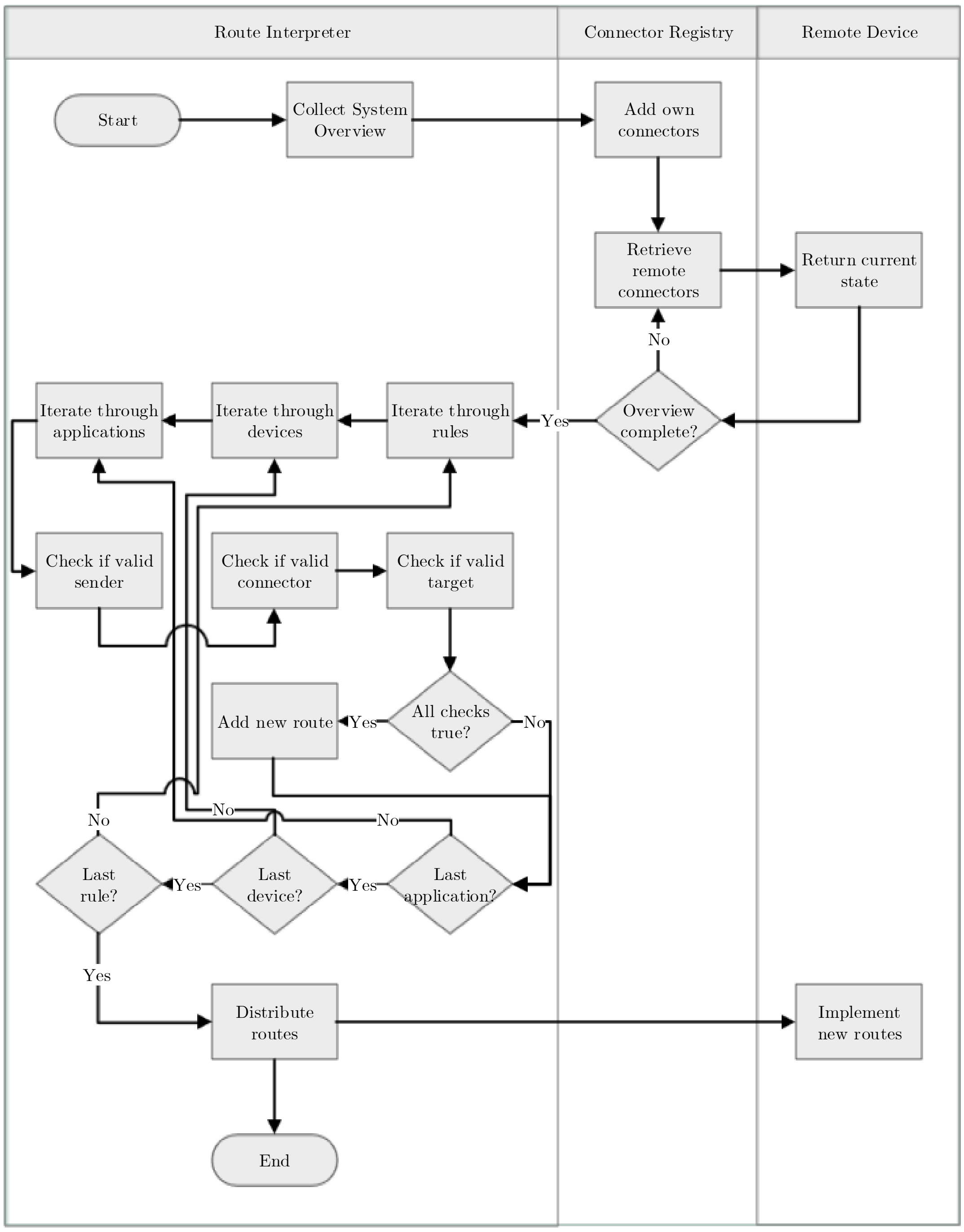


Figure 6.5.: The route interpreter first needs to obtain a complete overview of the pervasive system and afterwards evaluates each rule for all applications on each device. Lastly the resulting routes are distributed throughout the system.

set which contains the devices for which the routes are generated, and the system image with all devices which are needed to determine the targets of the routes. If only a partial reconfiguration is needed in the case of a small change in the system, the first set is substantially smaller. For a complete reconfiguration both sets of information will be equal.

The actual interpretation of the rules is executed per rule in the configuration. For each rule, the interpreter iterates through all devices and tries to apply each rule to each application of the specific device. During each iteration it first checks, if the application is a valid sender for the rule and if it has a connector which fits the rule. Afterwards, it determines all valid targets for the current route. If any of these three checks fail, this iteration is aborted directly to save on the computation and the interpreter continues with the next pair of application and rule. To check if the current application is valid as a sender, the interpreter compares the sender attribute of the rule with the ID of the application. If the tag is "ALL" or the ID is contained in the rule, the stored application properties are compared to the sender filters of the current rule. Multiple filters are interconnected by a logical "*and*". Therefore, for each filter, the interpreter looks for a valid property. If the application is valid as a sender for this rule, the interpreter iterates over all of its connectors. For each connector the interpreter checks, if it fits the data and message type of the rule and if it is an outgoing connector. In this case the application is able to send information via this connector according to the current rule. Thus, we are able to create the beginning of a new route. In the next step, the Rule Interpreter iterates again over all applications in the system to find valid targets. To qualify as a valid target the ID of the application has to match the receiver tag and the properties the receiver filters of the rule. This is similar to the validation of the sender. Additionally, it is checked, if the application has an ingoing connector with a matching data and message type for the current rule. If all these conditions are met, the application is added to the list of valid targets for the new route. After iterating over all applications the route is finalized by adding the context filter of the rule, if one was set by the administrator or user. The now finished route is added to a temporary list and the interpreter continues with the next application.

Formally, to define the needed information about the structure of the pervasive system let $A = \{a_1, ..., a_n\}$ be the set of all applications in the system and n is the total number of applications. A specific application a_j is defined by its ID. Further, for each application a_j exists a set $P_{a_j} = \left\{p_{1,a_j}, ..., p_{m,a_j}\right\}$ with all properties and the sets $C_{a_j} = \left\{c_{1,a_j}, ..., c_{l,a_j}\right\}$ with the connectors of the application. Also, m is the amount of properties and i is the amount of connectors for this application. Each property p_j is a vector (k, v) with k as the key and v as the value of the property, e.g., $p_1 = (group, student)$. The connectors c_j are defined as a vector (mt, dt, dir) where mt is the message type, dt the data type, and dir the ingoing or outgoing direction for this connector, e.g., $(chat, \backslash T, ingoing)$.

Additionally, each rule in the currently active configuration is defined by the sets $S = \{a \in A \mid \textit{Sender in rule}\}$ and $R = \{a \in A \mid \textit{Receiver in rule}\}$ which contain all senders and receivers defined in the current rule. Each sender/receiver is defined again by the application ID. For each rule we also define a set of filters $F = \{f_1, ..., f_p\}$ with p as the total amount of filters. The two subsets $F_S \subseteq F$ and $F_R \subseteq F$ contain the filters regarding the sender S and receiver R. A filter $f \in F$ is similar to the previously defined properties, a vector (k, v) containing a key and value. Further, each rule contains a vector msg, which is defined with (mt, dt, dir) exactly as the connectors c of the applications. If the rule is applied to possible source applications, the dir value of the msg is set to “outgoing” and for possible receivers to “ingoing”.

$$RT_S = \{a \in S \mid F_S = P_a\} \cap \{a \in S \mid msg \in C_a\} \tag{6.1}$$

$$RT_R = \{a \in R \mid F_R = P_a\} \cap \{a \in R \mid msg \in C_a\} \tag{6.2}$$

The expressions 6.1 and 6.2 show the evaluation of one rule in the configuration to generate the corresponding routes. The resulting set RT_S of the expression 6.1 contains all applications which are a valid sender for the current rule. This includes all applications where the set of properties P_{a_j} is equal to the sender filters F_S and where at least one connector in the set C_{a_j} exists that is equal to msg. Analogously, expression 6.2 returns the set RT_R with all possible targets. The evaluation is similar to the sender, but uses the corresponding set of receivers R and receiver filters F_R. If both resulting sets contain at least one application the algorithm is in the next step able to create the routes. All sender applications are updated with the future targets for their connector fitting the *msg* vector of the rule.

After iterating over all applications and rules, the calculated and temporarily saved routes need to be distributed to their according devices. Therefore, each device runs its own *Local Route Handler*, which receives routes from other devices and provides them to the *Route Controller* when sending information as discussed in Section 6.5. After the interpretation, each device is updated with its new routes. In the case of a complete reconfiguration it is told to drop all existing routes, while during an incremental change the new routes are appended. The process of distributing the routes to all devices is parallelized.

6.4. Configuration Distribution

The previous sections discussed how the information transmitted via the PerFlow Middleware is structured and how it is possible to create rules and interpret them as routes for each device. Until now, the interpretation is executed directly on the device, where the configuration change happens and all the other devices are updated with their future routes. To allow all peers to create new rules and reconfigure the system, the configuration containing the rules has to be distributed to all devices after each change. Therefore a *Configuration Manager* is introduced to the PerFlow Middleware. The PerFlow system offers two different strategies for the distribution, which will be discussed in the remainder of the section. First, a naive and simple approach for sending the configurations to all devices. Second, a complex approach with more fine grained control including the election of a leader and a consensus strategy. Therefore, the *Configuration Manager* is extended with a *Consensus Module*.

6.4.1. Naive Configuration Distribution

For handling and distribution of new and existing configurations a *Configuration Manager* is introduced in the PerFlow system. The main task of the manger is to store the configuration, offer it to the interpreter, and distribute it to all other peers in the system. While the first two tasks are similar in both the naive and consensus based approach, the distribution differs. Only the last valid configuration for the system is stored on each device to be accessed by the interpreter. Further, the manager takes care to only overwrite an outdated configuration if a new one is available to ensure that the system is consistently operational. To allow the Configuration Managers on different devices to communicate for coordination and exchange of configurations, they are designed as services of the underlying communication middleware with a fixed service ID.

For the distribution of the configurations the naive approach relies on the Transmission Control Protocol (TCP) based reliable data transfer offered by the underlying communication middleware. If the manager receives a new configuration, it asks for all available devices in the pervasive system and sends the configuration to them individually. There are no further evaluations involved checking for instance if the executing device is allowed to introduce a new configuration or if it is valid and leads to a correctly functioning system. Further, the configuration manager is also responsible for newly connected devices to being able to catch up with the already running system and retrieve the currently valid configuration. Therefore, in the naive approach, at the start of the system, the manager picks out one other peer in the system randomly and asks it for its current configuration. After receiving it, the manager stores it and the startup process of the PerFlow Middleware continues.

6.4.2. Leader and Consensus based Configuration Distribution

While the fast but very basic naive approach may be sufficient for many scenarios, there are use cases where a more fine grained control and more guarantees are welcome. In the case of the PerFlow system these are *access control*, *fault tolerance*, and *consistency*. To achieve the desired *access control*, it should be possible for administrators to define rights to steer who is able to reconfigure the pervasive system. When a new configuration is introduced during runtime, these rights need to be evaluated and a decision to reject or implement the configuration has to be made. The additional guarantees for *fault tolerance* and *consistency* can only be given if all peers in the pervasive system come to a uniform decision regarding the new configuration. Therefore, the *Consensus Module* is added to the *Configuration Manager* and a consensus protocol is introduced to coordinate the decision process between the different devices [155]. Such a protocol satisfies three properties, which help to ensure the aforementioned guarantees: termination, integrity, and agreement [39].

To design a consensus protocol for a distributed computing system, it is important to be able to replicate the data needed for the decision making across all devices. The underlying concept enabling a fault tolerant distribution of data is called state machine replication [174]. This concept is also the basis for many different consensus protocols like Paxos [106], Raft [145], or Viewstamp Replication Revisited [113]. For the PerFlow system the Raft protocol was chosen, which allows the replication of a state machine across different devices to ensure agreement between them. In comparison to other consensus protocols, especially Paxos, Raft offers a well-documented and accessible algorithm, which allows for an easy and lightweight integration to the middleware. Raft achieves a consensus in the distributed system via a leader-based approach. All peers in the system elect one leader, which is then solely responsible for making the decision and replicating the state to all others. While such a leader-based protocol is able to cope with $f < n/2$ faulty processes with n as the total number of processes, it is not possible to detect byzantine failures [107]. As the configurations in the PerFlow system are created by the user and not calculated, this poses no large disadvantage. There are already several implementations of consensus protocols like the Akamai Configuration Management System [175], Apache ZooKeeper [89], or Consul by HashiCorp [75]. As they all come with some drawbacks, especially regarding the size and flexibility of the system, they are not directly usable for the PerFlow Middleware. Therefore, the decision was made to base the configuration distribution directly on the Raft consensus protocol. While pervasive systems often consist of mobile devices that leave and join the system, we still encounter many stable devices e.g., the laptop of the lecturer or infrastructure devices like servers. Therefore, it is possible for a leader based consensus approach to elect a leader that has a possible high uptime and thus reduce the need for reelections. In such cases the overhead introduced by the elections is minimal while the actual distribution of the configurations does not differ from the naive approach.

The remainder of the section will first take a look on how the leader in the pervasive system is elected before talking about the design of the access control system and the actual configuration distribution. Afterwards, it is discussed how newly started instances in the system are able to catch up with the rest.

Leader election: To detect the absence of a leader, a heartbeat mechanism is introduced, the heartbeat is sent in regular intervals to all peers in the system. If one peer does not receive a message within a certain timeout, it triggers the process to elect a new leader. In the Raft protocol, the heartbeat and timeout interval are randomly chosen [145]. Thus, it can occur that the timeout is smaller and a new election is forced even though the current leader is still available. This is introduced intentionally to switch the leader regularly and to not only have one device with the overhead. For the PerFlow system, this behavior is not desired, as the constant new elections would introduce a larger overhead on the system compared to the load on the leader. Therefore, the heartbeat interval is fixed and the timeout is always set to twice that value. To further reduce the amount of new elections, the standard heartbeat is set to a high value of 400ms compared to the standard Raft algorithm, but it is adjustable by the administrator to the current use case.

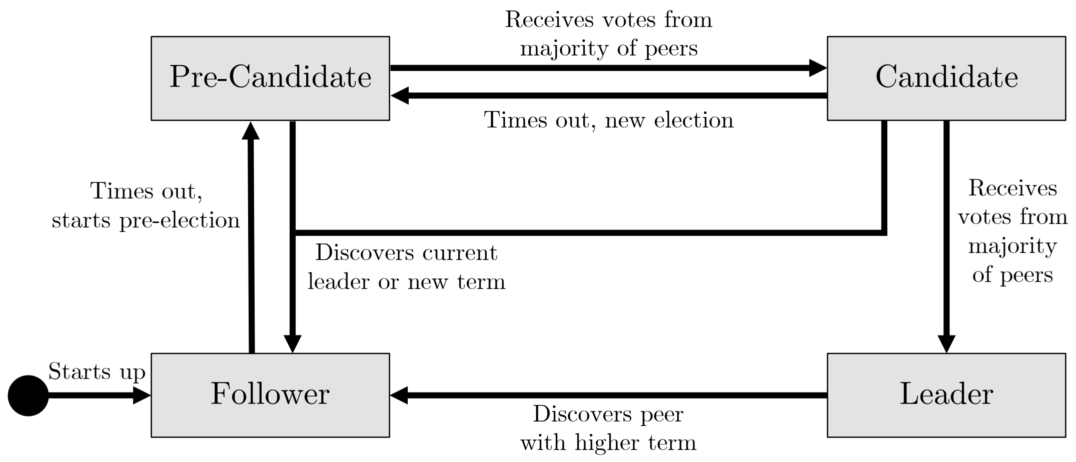

Receives votes from majority of peers
Pre-Candidate
Candidate
Times out, new election
Times out, starts pre-election
Discovers current leader or new term
Receives votes from majority of peers
Starts up
Follower
Leader
Discovers peer with higher term

Figure 6.6.: The different possible states a device can have in the pervasive system and how it is possible for the devices to transition between states. The model is an extension of the states introduced in the Raft protocol [145].

Figure 6.6 shows the different states a peer can have and the transitions between them. While the states of *follower*, *candidate*, and *leader* are directly adopted from the Raft algorithm, an additional *pre-candidate* state was introduced with the goal to minimize the amount of elections. This is based on the idea of a pre-election, which was amongst others mentioned by Ongaro for the Raft algorithm [144] or introduced by Junqueira *et al.* for ZooKeeper [100]. A comparable type of pre-elections was implemented and tested for the Raft algorithm by Ingo [93]. During normal operation, all devices except one *leader* are in the *follower* state. If one of these devices detects a timeout of the leader, it transitions into the *pre-candidate* state and issues a request for a pre-vote to all others. In the case that the other devices acknowledge the need for a new election (e.g., because they also detected a timeout), they answer positive to the request. If not enough devices respond positive or the *pre-candidate* receives a message from the current *leader* it thought was missing, the pre-election times out and the device transitions back into the *follower* state. As soon as a majority of the devices gave positive feedback, the *pre-candidate* advances to the *candidate* state. In the next step, the new *candidate* device announces the election together with information regarding itself containing its ID and the version of the last configuration it holds. All other devices decide the validity of the candidate based on this information and if a majority responds positive to the election, the device transitions to the *leader* state.

To specify the current state of the system, the additional concept of terms as logical clocks is introduced [105]. Each device in the system saves the current term. This value is increased each time the election of a new leader is started. If a device receives a request from another peer e.g., a new candidate starting a vote with a lower term than its own, it knows that this peer is outdated and rejects the request. Additionally, if a device currently in the leader state receives a request with a higher term value, a new leader must be present and it reverts back to the follower state.

Configuration distribution and access control: The main tasks of the leader are to check if a new configuration is valid and to distribute it to all available devices in the system. If any device in the system, no matter if it is in the follower or leader state, wants to introduce a new configuration to the system, it hands the configuration over to the current leader. Similar to the term concept, the configurations are assigned a version number which is increased by the leader upon distribution. If the leader receives a configuration with a lower version than the one stored locally, it is seen as outdated and the leader refuses the request to reconfigure the system. Second, the leader checks if the requesting device is actually allowed to introduce new configurations to the system. After making sure that both conditions are met, the leader distributes the new configuration by utilizing the reliable TCP communication of the underlying middleware. The process of actually distributing the configuration is similar to the naive approach mentioned before.

```
<configurationsettings>
  <right>
    <property>group</property>
    <value>teachers</value>
    <priority>2</priority>
  </right>
  <right>
    <property>group</property>
    <value>assistant</value>
    <priority>2</priority>
  </right>
  <right>
    <property>group</property>
    <value>students</value>
    <priority>1</priority>
  </right>
</configurationsettings>
```

Listing 6.3: Example XML representation of the rights used for the access control in the Consensus Module. Showing three different rights for lecturers, assistants, and students with different priorities.

To introduce access control to the system, administrators can provide an XML file with defining rights when deploying the system. During runtime, the current leader uses these rights to evaluate if the requesting device is allowed to reconfigure the system. Listing 6.3 shows an example with three different rights. Each right contains a property, value, and priority. The property and value match the properties saved in the Connector Registry and the priority tells the leader in which order to evaluate the rights. If a lower priority is set for one right, it is only evaluated if no match can be found for the rights with a higher priority. Additionally, if two rights with the same priority are present, they are both evaluated equally. The three rights provided by the administrator in the example first require the device to be in the group “teacher” or “assistant”. If none of the currently present devices matches these requirements, the right with the higher priority is evaluated and also devices in the group “student” are allowed to reconfigure the system. Thus, the system remains operational if no teacher or assistant is present at the moment.

Catch-up mechanism: To accommodate devices joining to the pervasive system or to allow leaving or failing devices to return at a later point in time, the configuration manager offers a mechanism to catch-up with the rest of the system. In the heartbeat messages, the leader includes the current term and the version number of the currently active configuration. Therefore, if a joining device receives a heartbeat, it does not only know who the leader is but also the status of the system. If the new device has the same version of the configuration as the leader, it updates the locally saved term and is finished. However, if the configuration version of the device is lower than the received number, it knows that it is outdated and contacts the leader to receive the updated configuration. Lastly, if the joining device was in the state of a leader before leaving or failing, it knows after comparing the term number that a new leader exists and falls back into the state of a follower.

6.5. Configuration Enforcement

The PerFlow Middleware allows developers to provide information about their applications and administrators or users have the ability to create rules to configure the information flow. These rules are then interpreted and specific routes for the information of each device are created. The rules and routes are then distributed throughout the system. Until now, the information flow in the system is not affected in any way and the applications are not able to communicate with each other so far. Thus, the following section we will discuss how the middleware steers the information flow. We first take a look on what developers need to do to enable their applications to send and receive information in the pervasive system. Afterwards, the information handling of the PerFlow Middleware is described and how it enforces the rules defined in the configuration.

6.5.1. Sending and Receiving Information

While PerFlow eases the process of sharing information in a pervasive system substantially, the developer still has to prepare the application for using the middleware. The preparation includes starting the middleware, registering the application and connectors, and setting up the application for sending and receiving information. Here, the overall goal of the middleware during this process is to aid the developer as much as possible and introduce simple to use interfaces. Therefore, we aim at hiding the complexity from the developer and position the PerFlow system between the application and communication middleware. Thus, the developer does not need knowledge about other devices and applications in the system. Developers also do not have to worry about who is interested in what information and how it can be distributed accordingly.

Figure 6.7 depicts the actions a developer needs to perform in order to being able to receive and send information via the PerFlow Middleware. In both cases, the application first needs to start up the middleware and register the application itself. The setup automatically starts the communication middleware, the PerFlow Middleware, and then initializes the Connector Registry and Local Route Handler. After everything is up and running, the developer registers his application with the Connector Registry. The properties of the application need to be provided here and the registry automatically assigns a unique ID. Afterwards, the developer is free to register any connector with the registry by providing the needed information as discussed in Section 6.2.1. This step is not necessary directly at the startup of the application. The developer is free to register new connectors during the applications runtime or remove already existing connectors if the functionalities behind them are no longer offered by the application.

While the process of starting the middleware and register the application and connectors is identical for all developers, we afterwards have to differentiate between providing and receiving information. In both cases we differ between active and passive connectors. To be able to send information, the developer first has to ask the middleware for an instance of the Message Proxy. This proxy is responsible for serializing the data and handing it over to the Route Controller of the middleware. Therefore, it offers a method for each currently supported

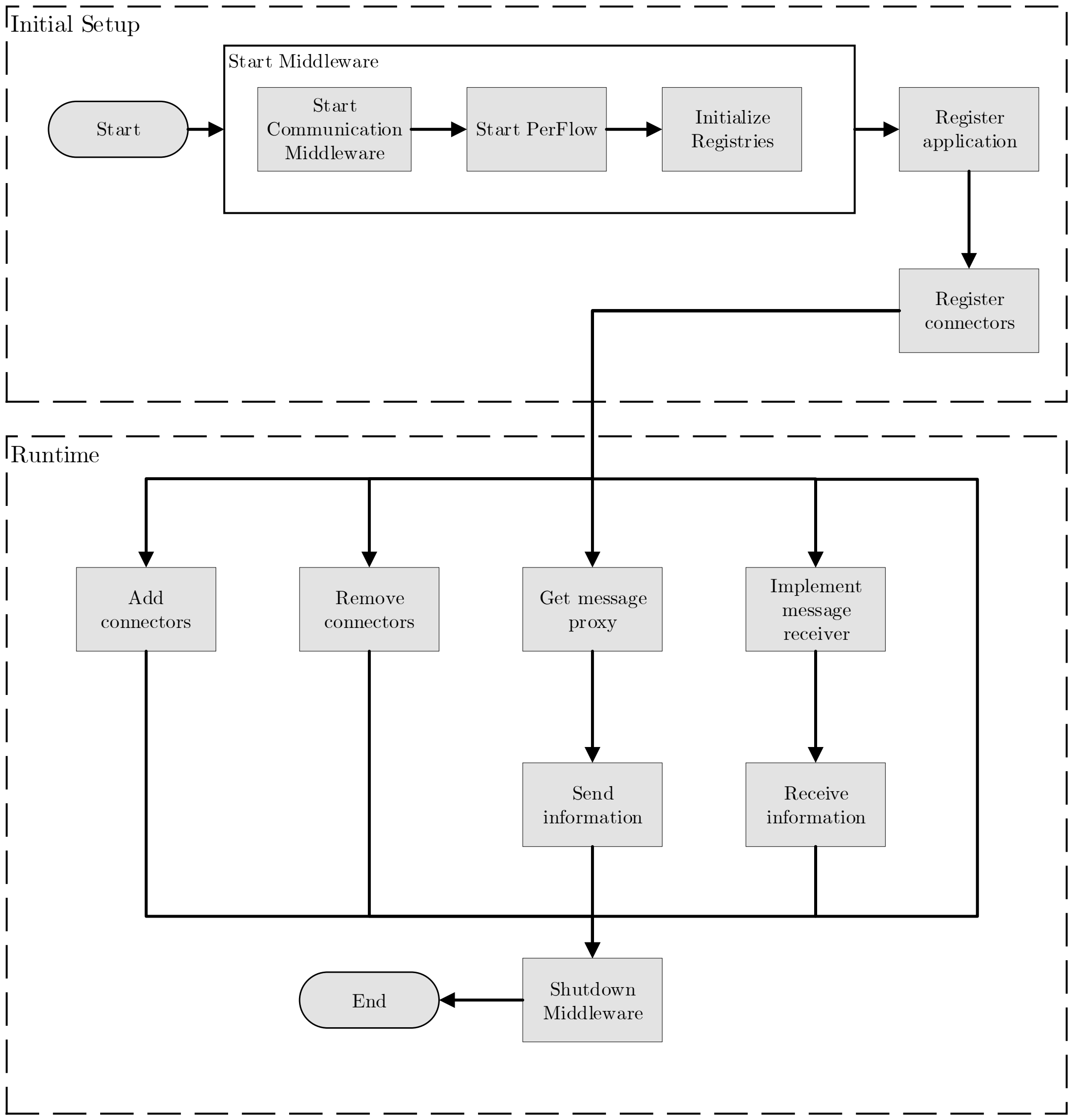


Figure 6.7.: The actions the developer has to take to initialize the middleware during the initial setup and the offered functionalities during runtime to change connectors and send or receive information.

data type. If the developer registered an active connector and the application wants to transmit data, the developer calls the regarding method and provide the message type of the information as a string. In the case of a passive connector, the developer implements an interface of the Message Proxy and returns the data when the PerFlow Middleware calls the corresponding method with the message type.

For receiving information via PerFlow, the developer uses the Message Receiver offered by the middleware. If a passive connector is used, the application again implements an interface offered by the receiver. This includes a method for every supported data type and developers need to override the methods they are interested in. The corresponding methods are then called upon receiving new information via the PerFlow Middleware. For an active connector, the developer needs to call the Message Receiver and ask for the data by providing the message type. The middleware then returns the last received information fitting the call. In both cases, the middleware is responsible for the deserialization of the data. Additionally, the application is provided with the message type and the ID of the sender.

The Message Proxy and Receiver are able to serialize all primitive data types and images. In addition to single values, it is also possible to transmit arrays. If the developer needs other objects to be handled by the middleware, it is possible to extend the proxy and receiver. First, the new custom object has to implement an interface offered by the middleware for serializable objects. In the implementation, the developer specifies how the object can be serialized and deserialized into primitive data types e.g., integers or byte arrays. Second, one method has to be added to each the proxy and receiver class handling the new object.

6.5.2. Information Handling

After looking at how applications can hand over information to and receive it from the middleware, the next step is to look at how the information is handled by the PerFlow Middleware. Therefore, we are discussing the complete way of the information from calling the message proxy at one application to getting the data at the message receiver at another application.

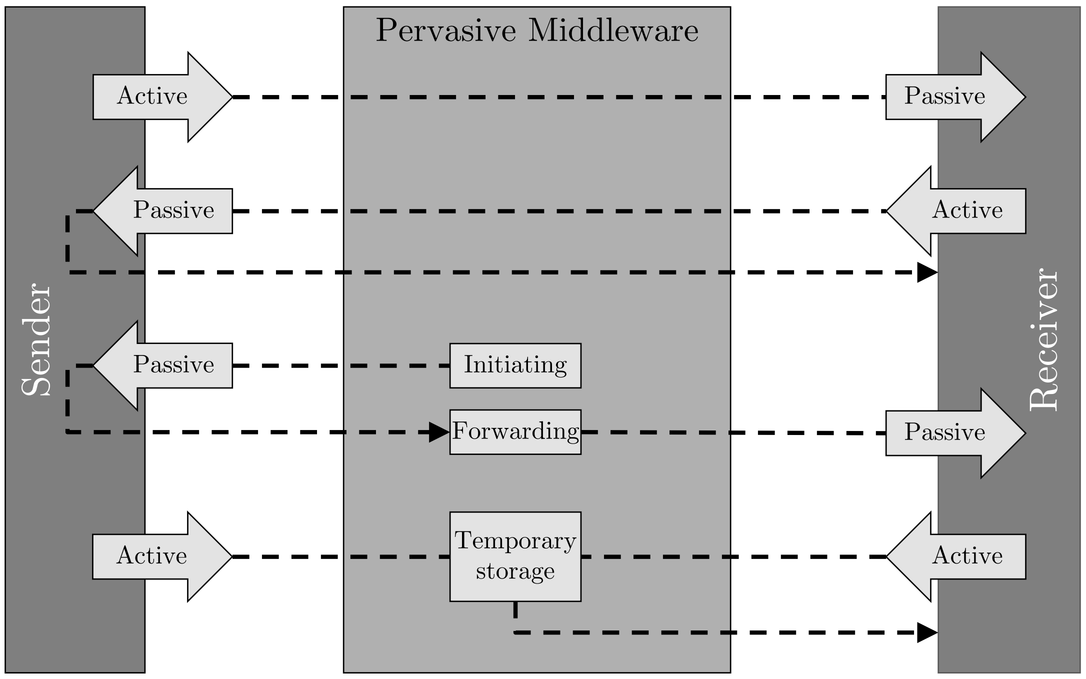


Figure 6.8.: The four different possible strategies for the transmission of information in the PerFlow Middleware depend on if the connector of the sender and receiver are defined as active.

For the transmission of information via the PerFlow Middleware we encounter four different strategies depicted in Figure 6.8. These strategies depend on the different possible combinations of active and passive senders or receivers and are based on the approach of Roth *et al.* [164]. In the case of an active sender it calls the Message Proxy as soon as it wants to transmit information. If the sender is passive, the middleware has to request the information from the sender. This happens either when it receives a request from an active receiver or after a regular interval, which can be defined by the administrator. For a passive receiver the middleware forwards the information to the application as soon as it receives it from the sender. In the case of an active receiver it pulls information from the middleware. The middleware then either sends a request to the sender if it is passive or forwards the temporarily stored last value received from an active sender.

After the Message Proxy received new information from the application, the PerFlow Middleware performs two tasks in parallel. On the one hand, it serializes the data as specified by the developer and prepares a message object.

This includes the actual data, the ID of the sender application, and the message and data type. On the other hand, the middleware determines the targets for this message. As the interpretation of the rules is already done by the interpreter, it is possible for the Route Controller to determine the targets for a message without a large computational effort or knowing the structure of the complete pervasive system. Therefore, the Route Controller iterates over all routes stored in the Local Route Handler of the device for this specific application. To determine if a route is applicable for this message, it compares the message and data type. It then composes a list of the application IDs for every valid target. In the next step, the proxy uses the underlying communication middleware to send out the message object to all targets. The transmission of the data is parallelized to increase the performance in large systems with many targets for every message. After receiving a new message via the communication middleware, the Message Receiver deserializes the transmitted information according to the data type which was communicated together with the message. The information is then handed over to the according methods of the Message Receiver interface implemented by the developer. When receiving new information, the application is given the actual data transmitted together with the ID of the sender and the message type.

6.6. PerFlow Tool

In the previous sections we discussed the PerFlow Middleware in detail and showed how it enables developers to implement information exchange for their applications. Additionally, PerFlow allows administrators and users to configure the flow of information in the complete pervasive system during runtime. Until now, they have to create their new configuration by defining rules in the JSON format. This may be suitable for administrators with a background in information technology. If we consider users like an art teacher or philosophy professor, we cannot expect them to learn JSON and the required configuration file format. To fulfill the requirement R_{NF4} for the usability of the system, we need to cater to all possible users in the pervasive system. To enable also users without IT experience to use all functionalities of the system, we are offering a visual scripting language combined with PerFlow Tool to create the rules graphically. The tool enables users to change the configuration of the system during runtime to fit the use case. Additionally, it is possible to create configuration beforehand, so that during runtime the user only needs to load the predefined configuration.

In the following section we will take a deeper look into the PerFlow Tool. Therefore, we first discuss the visual scripting language itself by defining all language elements and explaining how this enables the user to create the same rules as the JSON configuration. Afterwards, the tool itself is discussed by looking at the functionalities it offers to the user. Lastly, we will take a deeper look on how the PerFlow Tool is connected to the PerFlow Middleware. This includes receiving information about the structure of the pervasive system from the middleware and providing the newly created rules to it.

6.6.1. Visual Scripting Language

There are a multitude of different ways to design a visual alphabet discussed in recent years [38] and it is important to consider the use case [59]. The different approaches are often based on flowcharts [33], data flow [80], or events [24]. For the visual scripting language, which is the basis of the PerFlow Tool, we decided to use a data flow approach. It resembles best the flow of information in the pervasive system and gives a natural way to interact with the communication starting with the sender until the data reaches the receiver. Following, we will first take a look at the language itself and afterwards we will show how the different elements can be combined to create rules for PerFlow. In the following section we will demonstrate the design of the visual scripting language and the tool with mockups. Figure 7.2 in the implementation chapter shows a screenshot of the prototype of the PerFlow Tool with an example rule.

In Figure 6.9 the complete visual alphabet for the tool is shown. It contains six different elements in total, which will be discussed in more detail. The different elements have a common design language. They are all box shaped with the title of the element at the top and, if necessary, one or multiple parameters below. The parameters are mostly defined by drop-down and check boxes to minimize the possible errors a user could make by entering invalid information. Further, each element has one or two ports, which are used to connect several elements together. Input ports are always on the left and output ports on the right side

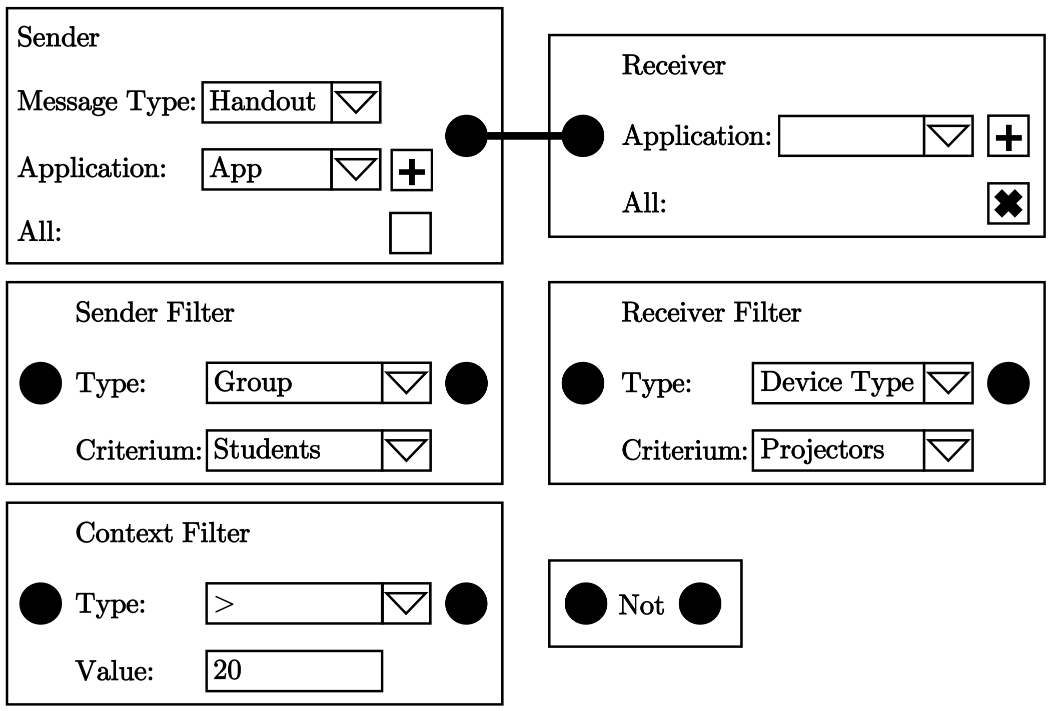


Figure 6.9.: The six different visual scripting elements available to the user for creating new rules to influence the information flow in the system.

of the element. Therefore, a route is always designed from left to right, starting with a sender and ending with a receiver.

Sender and Receiver Nodes: In both cases it is important to define which applications should be affected by the rule. Therefore, the nodes offer a parameter to choose a specific application or a check box to specify all applications as possible senders or receivers. Additionally, the nodes offer a button with a "+"-sign allowing to add further drop-down boxes to select more applications. This allows the users to create groups of applications with just one node. If the check box for all applications is chosen, the drop-down selection for single applications is disabled. Following, the user has the possibility to narrow down the selection of applications with filter nodes. Additionally, the sender node also has a parameter for the message type for this rule. The user is required to choose the type and it then applies for the complete rule started by the node.

Filter Nodes: The visual alphabet contains a node for each of the three different filters supported by the PerFlow Middleware. For the Sender and Receiver Filter the visual scripting element offers drop-down boxes for the type and criterium of the filter. These correspond with the properties of the applications, with the type as the key, and the criterium as the value of a property. The last filter is for the context. Here we offer a drop-down box for the type and a text-field for the actual value. The value is then applied to the data of a message with the type as an operator.

Not Node: Finally, the visual scripting alphabet also offers a Not element. This can be chained in front of Sender and Receiver Nodes and negate their filter. Thus, instead of, for instance, sending the information to everyone in the group "student", the user can define to send it to everyone but applications in this group. Therefore, it is not necessary to add a high number of properties the user wants to use as a filter, if he only wants to exclude some of them. Additionally, it allows to check if an applications satisfies one filter but not another, which would otherwise not be possible.

To create a rule, the user has to always start with one or multiple Sender Nodes and finish with at least one Receiver Node. In between it is possible to refine the route with filters as fine grained as needed. The nodes are interconnected by their input and output ports, always connecting the output of one node to the input of another. Further, each port can have several connections. This allows, for instance, to add multiple receivers to the end of a rule or apply several filters. When a rule contains multiple filters of the same type, they can be added in sequence or parallel. If they are in a sequence, they are later evaluated as connected by a logical "and", while parallel filters are interpreted using a logical "or". Due to the different possibilities to connect the nodes, users are able to create complex rules with only a low number of different elements. Thus, it is easy to learn the visual scripting language and minimizes possible errors by limiting the users' possibilities to only valid parameters and combinations of nodes for their rules.

Figure 6.10 shows an example rule demonstrating how the elements introduced beforehand can be combined to configure a pervasive system. The configuration consists of one rule with only three visual scripting elements showing how it is

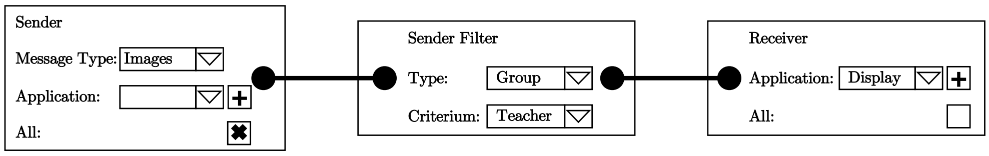


Figure 6.10.: A simple example rule with a specific application as sender and a filter for the senders.

possible to give one group access to a specific service. In this case all applications belonging to the group teacher are enabled to send images to a display service.

Following, we will introduce four different examples of rules created with the discussed visual alphabet and how they are evaluated by the system. Similar to the rule interpretation discussed in Section 6.3.3 we define the set $A = \{a_1, ..., a_n\}$ as the set of all applications in the system. Also, $P_{a_j} = \left\{p_{1,a_j}, ..., p_{m,a_j}\right\}$ is the set of all properties and $C_{a_j} = \left\{c_{1,a_j}, ..., c_{l,a_j}\right\}$ is the set of all connectors for the application a_j. By means of the examples we will show how the different visual scripting elements and their attributes relate to the information saved in the Connector Registry. For all examples we use the same message *msg* defined as a vector (mt, dt, dir), where the message type mt and data type dt is selected by the user as an attribute in the Sender Node. The direction *dir* is set to “outgoing”, when the senders are evaluated and “ingoing” for the receivers.

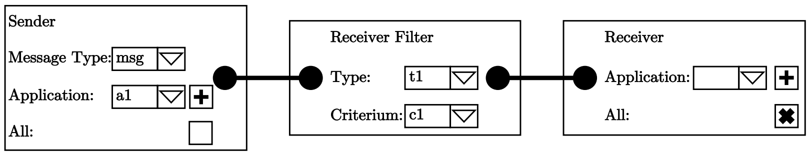


Figure 6.11.: A simple example rule with a specific application as sender and a filter for the receivers.

Figure 6.11 shows a very simple rule defining that one specific application is allowed to share information with a group of applications. Therefore, the possible senders are defined by the ID of the application with $a_1 \in A$, which also needs to satisfy $msg \in C_{a_1}$. The filter applied to the receiver is defined as a vector containing the type and criterium $(f_1 = (t_1, c_1))$. The set of possible receivers contains $Rec = \{a \in A \mid f_1 \in P_a \wedge msg \in C_a\}$.

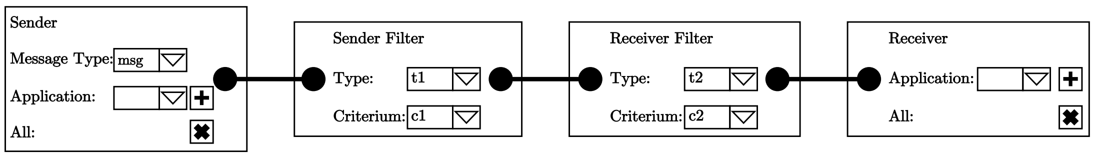


Figure 6.12.: An example rule with filters for sender and receiver.

In Figure 6.12 a more complex example with a filter each for the sender and receiver is shown. In this case the user did not choose a specific application with its ID for the sender or receiver. Instead, all applications that are able to satisfy the corresponding filters are included. The Sender Filter is defined as the vector $f_1 = (t_1, c_1)$ and the Receiver Filter as $f_2 = (t_2, c_2)$. Thus, the set of possible senders is defined as $Sen = \{a \in A \mid f_1 \in P_a \wedge msg \in C_a\}$ and the set of receivers as $Rec = \{a \in A \mid f_2 \in P_a \wedge msg \in C_a\}$.

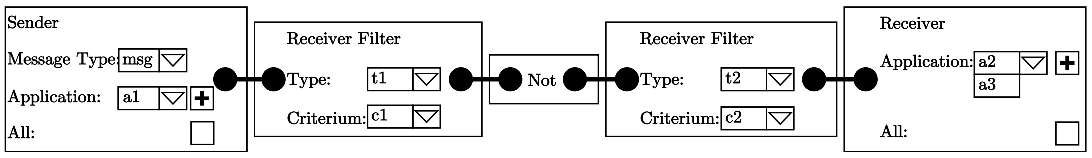


Figure 6.13.: Rule showing how several filters can be chained.

The next example in Figure 6.13 shows the possibility to chain several visual scripting elements to create more complex constraints for the sender or receiver influenced by the rule. In this case two Receiver Filters are chained and therefore combined with a logical “and” for the evaluation. Additionally, one of the two filters is negated with a Not Node. The two filters are again defined as the vectors $f_1 = (t_1, c_1)$ and $f_2 = (t_2, c_2)$. In this example the sender is directly specified with its ID $a_1 \in A$ and needs also to satisfy the condition $msg \in C_{a_1}$. The receivers need matching properties for the filter f_1 but not for f_2. Thus, the resulting set of valid receivers is defined as $Rec = \{a \in A \mid (a = a_2 \vee a = a_3) \wedge f_1 \in P_a \wedge f_2 \notin P_a \wedge msg \in C_a\}$ as the set of receivers for this rule.

The last Figure 6.14 contains a total of three rules serving as an example for parallel rules as well as parallel nodes in one single rule. In this instance, we showcase the flexibility of the visual scripting language, where the two rules above lead to the same result as the third rule. Users are able to create rules in different

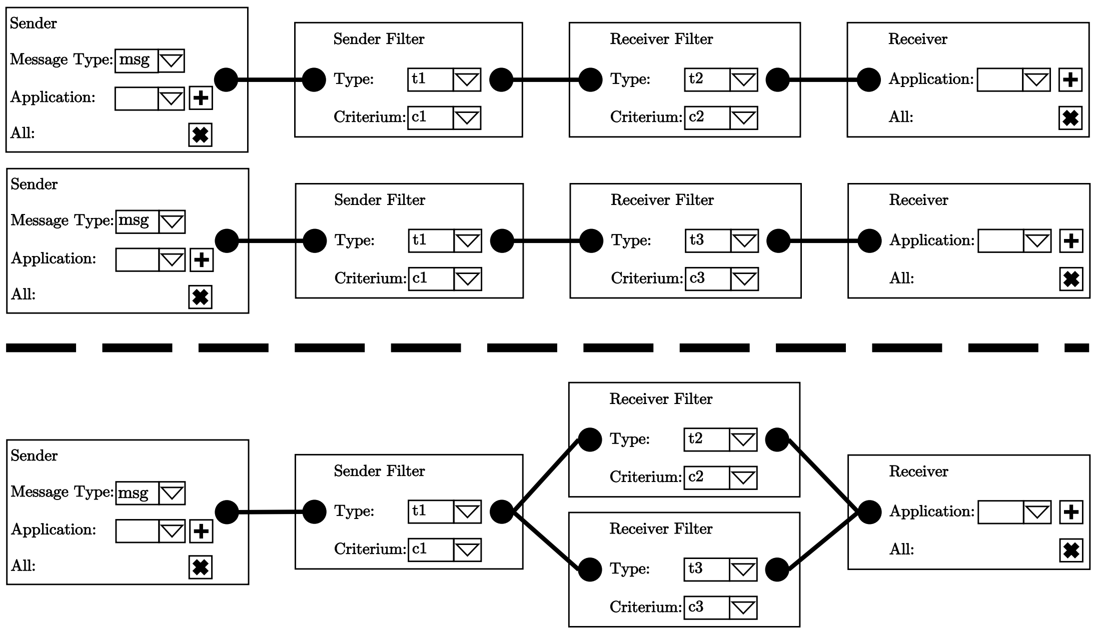


Figure 6.14.: A rule with a filter in parallel. The two rules above lead to the same result as the one rule with two parallel filters below.

ways according to their experience and understanding of the system. While some users may find the top example easier to understand, as it is more descriptive, the rule below uses less nodes and an experienced user may be able to create the rules faster. Again the filters are defined as vectors with $f_1 = (t_1, c_1)$ being the Sender Filter and $f_2 = (t_2, c_2)$ and $f_3 = (t_3, c_3)$ as the Receiver Filters. In both cases the senders have to satisfy the filter f_1 and the receivers either f_2 *or* f_3. Formally, this leads to a set of senders with $Sen = \{a \in A \mid f_1 \in P_a \wedge msg \in C_a\}$. Further, the receivers contain $Rec = \{a \in A \mid (f_2 \in P_a \vee f_3 \in P_a) \wedge msg \in C_a\}$.

The examples given above showcase just some of the different possibilities to combine the visual scripting elements to create rules for the information flow in the PerFlow system. Additionally, it is possible to create an unlimited number of different rules, which allows users to reconfigure the complete system as needed to fit their current use case. Further, users are able to achieve the same result in different ways fitting their current experience level with the offered system. The visual scripting language is also designed to prevent users from making syntactical mistakes by offering only attributes in drop-down boxes that will lead to a valid combination of nodes.

6.6.2. Visual Scripting Tool

For an easy to use system it is not only important to provide an understandable and clear visual scripting language. To create the rules, the users also need a tool which provides them access to the language in a simple way. Therefore, we created the PerFlow Tool, a simple-to-use application connected to the PerFlow Middleware for the easy creation of rules. With consideration of different user interface (UI) guidelines [9, 60, 123], the goal is to create an application which is focused and intuitive to use, but still very capable regarding the task of creating new rules. Moreover, it is also important to give the user feedback on his actions and the feeling that the application is not interfering during the reconfiguration process.

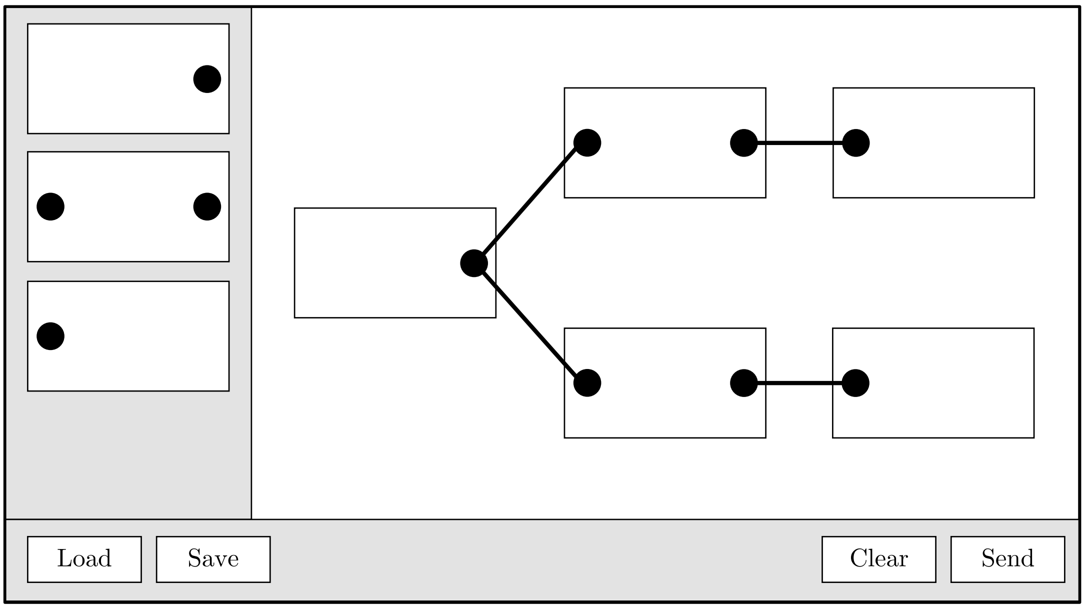


Figure 6.15.: The mockup for the PerFlow Tool with the list of visual scripting elements on the right, the menu on the bottom, and the canvas to create the rules in the middle.

In Figure 6.15 a mockup of the PerFlow Tool is shown. The user interface is divided into three main parts. The left side of the tool presents the user a list of all element contained in the the visual scripting language. The bottom shows the main menu for the user and in the middle we have the canvas for the creation of new rules. In the menu we have four functionalities for the user: loading a predefined configuration, saving the current configuration for later use, clearing the canvas, and sending the created rules to the PerFlow Middleware. With the loading and saving functionalities the user is able to predefine rules and load them during runtime if needed. Thus, a lecturer for instance can take his time to create the configuration when he prepares the lecture and can change quickly between different configurations while the lecture is going on. To create rules, the user can add new elements from the list on the left side via drag-and-drop to the canvas and position them freely. The relative positioning of the elements is not relevant for the syntax and does not influence the created routes. After adding an element, the user has to define the parameters via the drop-down, check, and text boxes contained in the element. Further, each of the parameters has to be filled out by the user and in the case of a missing parameter, an error message is presented to the user. If a parameter is not needed, the corresponding box is inactive and grayed out. Therefore, the user is prevented from creating invalid routes.

To create a rule out of these single elements, they have to be combined starting with a sender and ending with a receiver. A connection can be created between the output port of one element and the input port of another. To create this connection, the user simply needs to click on the two ports. In the same way it is possible to create a multitude of rules on the same canvas. As soon as the user is finished with the creation of all wanted rules for the complete system, he can send the new configuration to the PerFlow Middleware via the function offered in the menu.

6.6.3. Connection to PerFlow Middleware

The PerFlow Tool is not needed by every user in the pervasive system, as many of them are only using the services and may not even have the permission to reconfigure the pervasive system. In the case of our smart classroom example only the lecturer would have the right to create new rules for the system. To minimize disturbances during the lecture, students are only allowed to use the pervasive system as intended by the lecturer. Therefore, the PerFlow Tool is implemented as a standalone application and is not integrated directly into the middleware. Thus, we need to provide the tool with information about the devices in the pervasive system, their running applications and connectors, and the currently active configuration for the information flow. Also, in the case of a reconfiguration the new set of rules has to be handed over to the middleware.

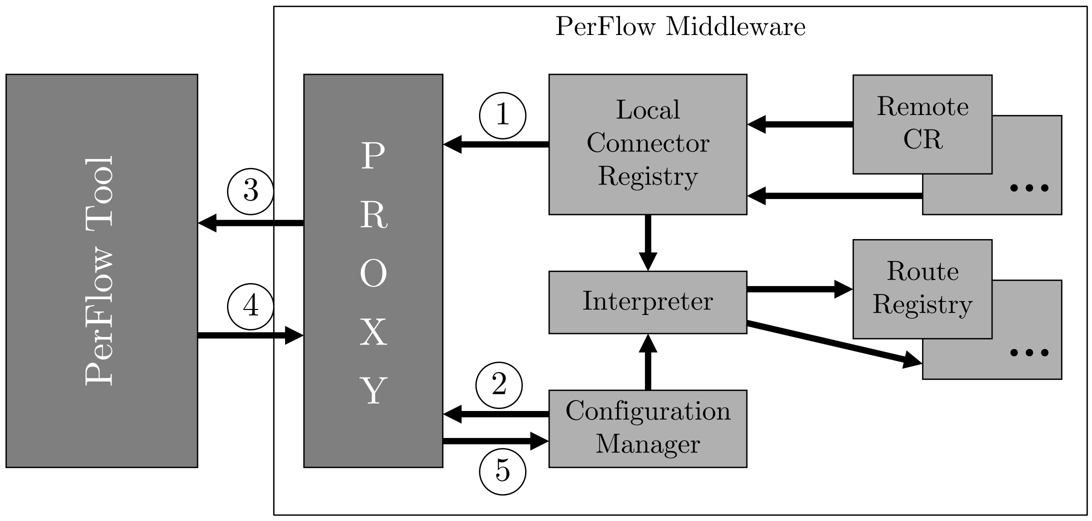


Figure 6.16.: Showing the communication between the PerFlow Middleware and PerFlow Tool utilizing a proxy for the visual scripting tool to retrieve information about the pervasive system (1)(2), forward it to the tool (3), receive the new configuration (4), and hand it over to the configuration manager (5).

To connect the PerFlow Tool and PerFlow Middleware, a small proxy application running as part of the middleware is developed. It is automatically started together with the visual scripting tool. Figure 6.16 shows the communication between the middleware proxy and the PerFlow Tool. On startup of the tool it requests the complete list of devices with their applications and connectors. To provide this information, the proxy calls the connector registries on all devices and collects the information (1). Additionally, the visual scripting tool asks for the current configuration, which the proxy automatically receives at the startup of the middleware (2). In the next step the combined information is handed over to the tool (3). Both of these steps are also executed if the PerFlow Tool is already up and running and the user requests the updated information by clicking on the load button.

After the user has finished with the creation of the new rules and chooses to send them to the middleware, they are handed over to the proxy (4). In this case PerFlow Tool creates a save file in an XML format. This includes the

```xml
<Canvas>
  <Elements>
    <Sender GUID="node-01" Top="271.0" Left="106.9" MessageType="msg"
        All="true"/>
    <Receiver GUID="node-04" Top="307.0" Left="1039.0" All="true"/>
    <SenderFilter GUID="node-02" Top="306.2" Left="377.0">
      <Property>
        <Type>t1</Type>
        <Criterium>c1</Criterium>
      </Property>
    </SenderFilter>
    <ReceiverFilter GUID="node-03" Top="306.0" Left="705.9">
      <Property>
        <Type>t2</Type>
        <Criterium>c2</Criterium>
      </Property>
    </ReceiverFilter>
  </Elements>
  <Connectors>
    <Connector StartElement="node-01" EndElement="node-02"/>
    <Connector StartElement="node-02" EndElement="node-03"/>
    <Connector StartElement="node-03" EndElement="node-04"/>
  </Connectors>
</Canvas>
```

Listing 6.4: XML representation of visual scripting elements for the second example route shown in Figure 6.12. It contains the properties for each visual scripting element, the connection between elements, and the coordinates on the canvas.

information about all elements placed on the canvas, their properties, and how the elements are connected. Additionally, the tool also saves the coordinates of all elements on the canvas. While this is not relevant for the middleware, it is needed if the configuration is later loaded again by any PerFlow Tool to being able to recreate the visual scripting elements for the user. Listing 6.4 shows the representation of the second example route seen in Figure 6.12. First, it contains all elements placed on the canvas together with the defined attributes. Each element has a unique GUID assigned by the PerFlow Tool and is saved together with the coordinate for its top left corner to be able to recreate the layout if it is loaded again. For the filters the XML also saves the selected properties.

Second, the file contains all connectors created between two elements. Each connector is defined by its start and end element. In the next step the PerFlow Tool transmits the XML file to the proxy. The proxy application then has the responsibility to translate the XML representation of the visual scripting elements into actual rules. Therefore, it creates a directed graph [20] with the information given in the XML file. The visual scripting elements are used as the vertices of the graph and the connectors as the edges directed from the start to the end element. In the next step this graph is used to generate the rules by using the Dijkstra shortest path algorithm [44] to find all paths starting with a sender and ending with a receiver. It then creates the configuration in the JSON format discussed in 6.3.1 and hands it over to the middleware to distribute the new rules in the system (5).

6.7. PerFlow Virtual Extension

To additionally enable remote users to participate in the pervasive system, we are enhancing the middleware with a virtual environment. In our scenario use case of a smart classroom this would for instance allow students abroad to take part in the lecture by joining it in a virtual classroom. The goal here is to not only give the users the same content, e.g., in the form of the lecturer slides, but to provide an experience as close as possible to participating in the real world. This could go as far as using virtual reality devices [184], like the Oculus Rift [51] or HTC Vive [85], to fully immerse in the virtual experience. Therefore, developers should have the possibility to translate the services and applications they developed for the physical smart environment directly to the virtual one. Thus, they need access to the same middleware functionalities in both cases and the pervasive system should handle all devices and applications in the same way no matter if they are physical or virtual.

The following section will first discuss how the virtual environment itself is designed and what possibilities it offers to developers and users. Afterwards, we will take a closer look on the connection of this environment to the PerFlow Middleware and how it communicates with devices and applications in the physical smart environment.

6.7.1. The Virtual Environment

The virtual environment is designed using a game engine as a middleware for the creation of three dimensional virtual environments [65]. The game engine does not only support the developers with the rendering of the environment, but also offers support for user input, networking, audio output and input, animation, or artificial intelligence. Due to this wide range of possibilities, game engines got the quasi standard in recent, not only for the development of video games, but also for other virtual environments used in areas like architecture [92], museums [110], or training and education [179] [199]. Therefore, many developers are already familiar with them or even completely rely on game engines for the development of their virtual environments. Additionally, many game engines are already providing support for virtual reality devices like the Oculus Rift directly out of the box. Thus, the decision was made to use an available game engine as the basis for the PerFlow Virtual Extension.

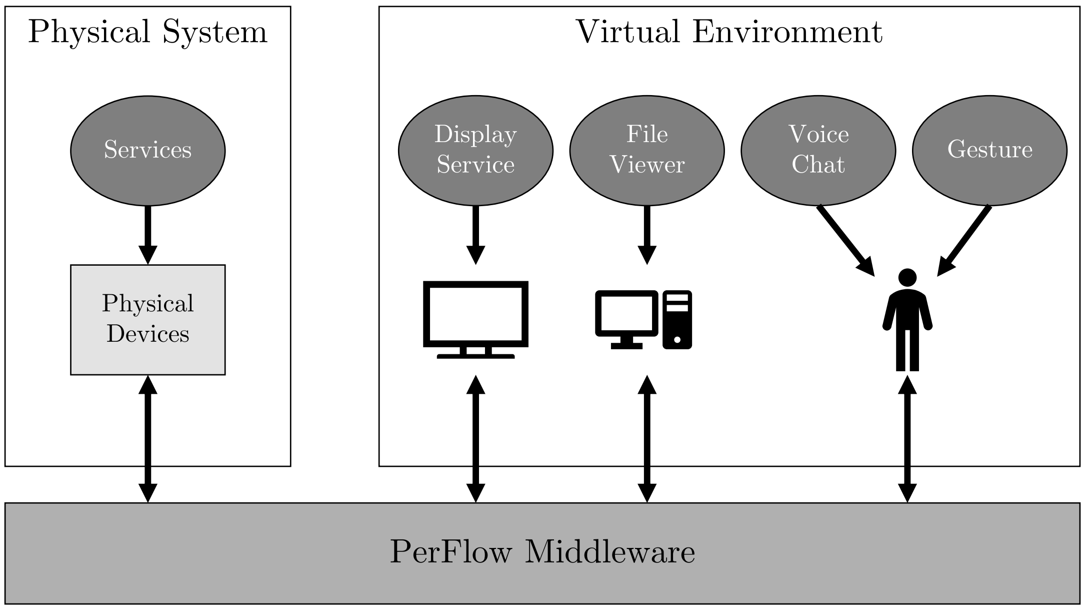


Figure 6.17.: The system model with the physical pervasive system and virtual environment connected by the PerFlow Middleware. In the virtual environment applications may run on virtual devices, avatars, or even directly in the virtual environment without a viewable representation.

The architecture of the virtual environment, as seen in Figure 6.17, allows for multiple virtual devices running in the same instance of the virtual environment. A device in this case could be the virtual representation of a physical device, e.g., a display at the front of the classroom or the computers used by the students. Further, it is also possible to develop applications or services which have no visual representation like a voice chat or gestures performed by students. Each of these devices and applications uses the Virtual Broker located in the virtual environment for the communication with the middleware. The broker offers the functionalities of the PerFlow Middleware to these peers. Therefore, the virtual devices can register their applications and connectors in the same way as physical devices. To send information, the developer calls the Virtual Broker which forwards it to the middelware. Further, for receiving information the virtual devices need to implement an interface offered by the broker. This procedure is similar to the Message Proxy and Receiver discussed in Section 6.5. The main goal of the Virtual Broker is to relay the method calls to the Virtual Controller located outside of the virtual environment. Incoming information has to be assigned to the correct virtual device and then distributed by the broker. All further logic and the communication with the actual PerFlow Middleware is left for the Virtual Controller.

6.7.2. Connection to PerFlow Middleware

The Virtual Broker offers the middleware functionalities to all applications inside a virtual environment and communicates via the Virtual Controller with the PerFlow Middleware. Following, we will first discuss the communication between broker and controller, which is realized via a local TCP connection. Afterwards, the connection of the controller to the PerFlow Middleware is described including how the information flow is relayed from virtual devices to the PerFlow system.

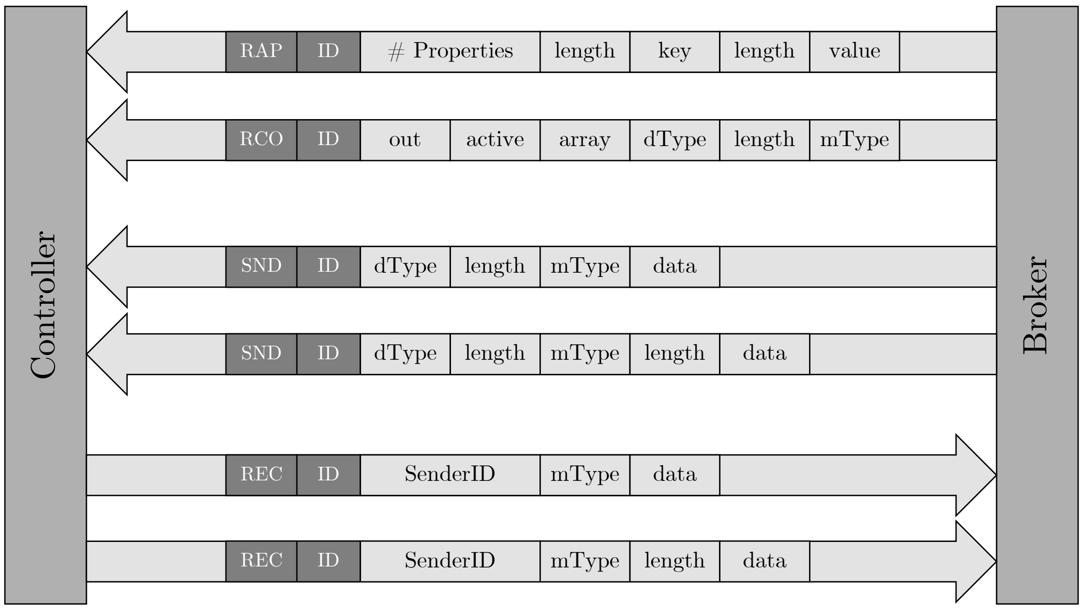


Figure 6.18.: The complete protocol for the communication between the Virtual Controller and Broker showing all messages with their transmitted information. The header for each message is highlighted in dark gray.

In Figure 6.18 the protocol used between broker and controller is shown in more detail. Upon the registration of a new virtual application, the broker assigns it a unique ID, which allows the controller and broker to address the correct virtual application in all further communication. Together with the request type the ID forms the following header for each message between the Virtual Broker and Controller:

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

The protocol defines four different requests: Registering an application (RAP), registering a connection (RCO), sending (SND), and receiving information (REC). The payload differs widely for each of the four messages. While registering an application the properties have to be provided to the middleware. Therefore, the protocol first expects the *number of properties* and then each property with its *key* and *value*. As both values are strings with an dynamic size, the broker first has to communicate the *length* before sending the data. For the registration of a new connector the broker sends the information for *active* and *in/out* as a single bit, where the 1 represents *active* and *out*. Afterwards, a byte representing the *data type* of this connector is transmitted, which is needed for the correct serialization. Lastly, the *length* for the following *message type*, which is represented as a string of dynamic length, is transmitted.

For sending information the protocol first expects the *data type* and *message type* fitting the registered connector. The transmission of the actual *data* differs between primitive data types with a fixed size and strings, arrays, or custom objects of variable size, which additionally need a *length*. When receiving data the protocol is similar and also transmits the *data type*, *message type*, and actual *data* in the same way as when sending information. Additionally, while receiving information the protocol also expects the *sender ID*, which is handed over to the virtual application.

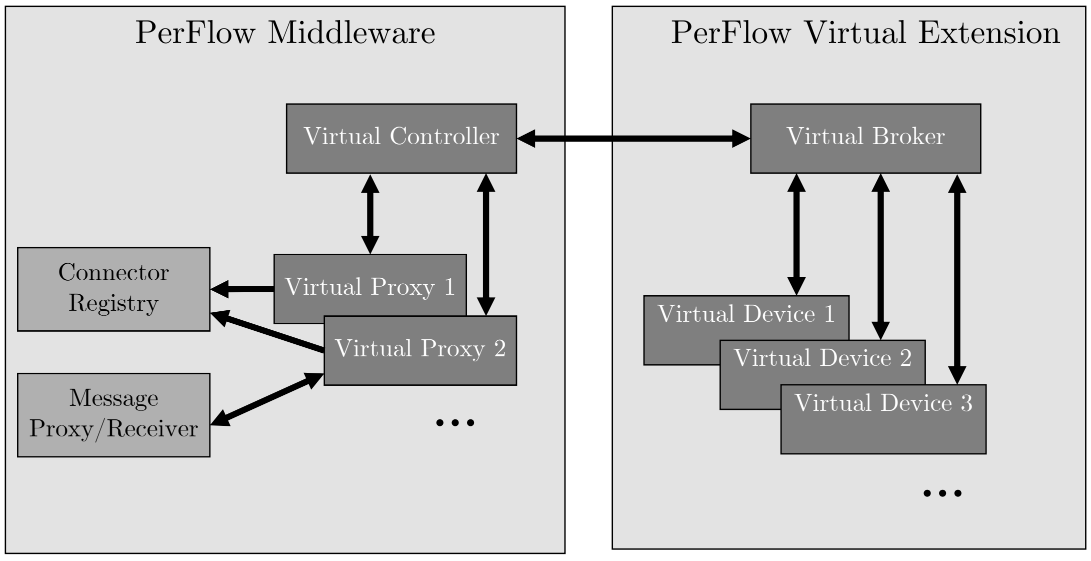


Figure 6.19.: Architecture describing the communication between the Virtual Controller located at the PerFlow Middleware and the Virtual Broker as a part of the PerFlow Virtual Extension. The communication with the middleware is handled by the Virtual Proxies.

The main task of the Virtual Controller is to orchestrate the different virtual devices and assign the correct information flow to each. The architecture in Figure 6.19 shows the information flow between applications in the virtual environment and in the physical pervasive system and how they utilize the Virtual Broker and Controller. After the broker asks for the registration of a new application, the controller starts a new instance of the Virtual Proxy and registers it at the PerFlow Middleware. Thus, there may run multiple proxy simultaneously on only one device for the same virtual environment. This allows the controller to map the incoming and outgoing information flow to the correct virtual application and communicate it accordingly to the Virtual Broker. These proxy act like an application running on a physical device in the pervasive system and are treated in the same way by the middleware. They register connectors on behalf of the corresponding virtual device and send the information handed over by the controller to the Message Proxy of the PerFlow Middleware. If a proxy receives information, it forwards it to the Virtual Controller, which then maps it to the correct virtual device and sends it to the broker. Further, if a device in the virtual environment is shut down, the controller is notified by the broker and also terminates the corresponding Virtual Proxy.

In the case of the virtual environment extension of the PerFlow Middleware, the serialization and deserialization of the transmitted data is split into two parts. First, the Virtual Proxy is responsible for the serialization of information sent to the middleware and the deserialization of data received from it. This is achieved by using the Message Proxy and Receiver as described in Section 6.5.1. Second, the messages also have to be serialized for the transmission between the Virtual Controller outside and the Broker inside the virtual environment. As we rely on a game engine for the virtual environment, we are also depending on the tools, libraries, and programming languages offered by it. Therefore, we are not able to directly use the data types supported by the PerFlow Middleware. If a virtual device is sending information, the Virtual Broker has to serialize the data and the Controller is responsible for the deserialization and vice versa for receiving information from the middleware. The PerFlow Virtual Extension supports, like the PerFlow Middleware itself, all basic primitive data types, as well as strings and images. Developers are able to introduce new custom objects to be sent and received by their applications. To achieve this, they have to extend the PerFlow Middleware as described in Section 6.5.1. If the developer also intends to use the newly added objects with virtual applications inside the PerFlow Virtual Extension, they additionally have to add the serialization methods to the Virtual Controller and Broker.

6.8. Summary

This chapter presented the design for PerFlow, a runtime configurable pervasive middleware combined with an virtual extension. The PerFlow Middleware supports the communication between applications by enabling developers to define what information their applications are able to send and receive. The actual transmission of the data is then handled by the middleware. For defining the connection between applications the configurations can be either created in JSON or by using visual scripting the PerFlow Tool. Therefore, we designed a flowchart based visual language and extended the middleware with a visual scripting tool. Newly introduced configurations are in the next step interpreted by the PerFlow Middleware and the resulting routes are distributed throughout the systems. For the distribution of these configurations we introduced two different approaches. First, a naive approach without any control of who is allowed to change the configuration. Second, a consensus algorithm based on Raft [145]. Lastly, PerFlow Virtual Extension was introduced allowing developers to implement pervasive applications and services in a virtual environment. To achieve this a proxy was designed to give developers using a game engine for their 3D environment access to middleware functionalities. In the next chapter the implementation of the PerFlow prototype is introduced.

7. Prototype Implementation

In the previous chapter the design for the PerFlow system was discussed in detail. The following chapter presents the prototypical implementation of the design. This prototype will later serve as the foundation for the evaluation. The three main artifacts of the prototype are the pervasive middleware, the visual scripting tool, and the virtual environment. These are implemented as close to the design introduced in Chapter 6 as possible and together they form the complete PerFlow system. Due to the similarity with the design, we will not discuss the complete architecture again in the following chapter and only focus on specific implementation details relevant to the prototype.

The remainder of the chapter is structured as follows: First, the implementation of the PerFlow Middleware is discussed. Hence, the first section looks at the underlying communication middleware and how it enables the different parts of the PerFlow Middleware to communicate with each other. Second, the PerFlow Tool is presented by giving details about the possible platforms for the tool, the implementation of the graphical user interface, and how the connection to the middleware is achieved. Third, the implications of the chosen game engine on the implementation of the PerFlow Virtual Extension is discussed. This includes a description of the connection from the virtual environment to the PerFlow Middleware.

7.1. Implementation of the PerFlow Middleware

The prototype of the PerFlow Middleware is implemented using the Java Platform, Standard Edition 9 [147]. For handling the rules in the JSON format the library Google Gson [61] is used. If a new set of rules is received, it is forwarded to the Gson handler, which deserializes the JSON file and creates Java Objects. The result is one main object containing a list with all rules. Each of the rules has the necessary information, as described in Section 6.3.1, saved as attributes.

Additionally, a route also contains a list of context filter objects, each containing the operator and value. Similarly, the Simple API for XML (SAX) parser [148] is used for parsing the XML configuration for the rights management of the Consensus Module described in Section 6.4.

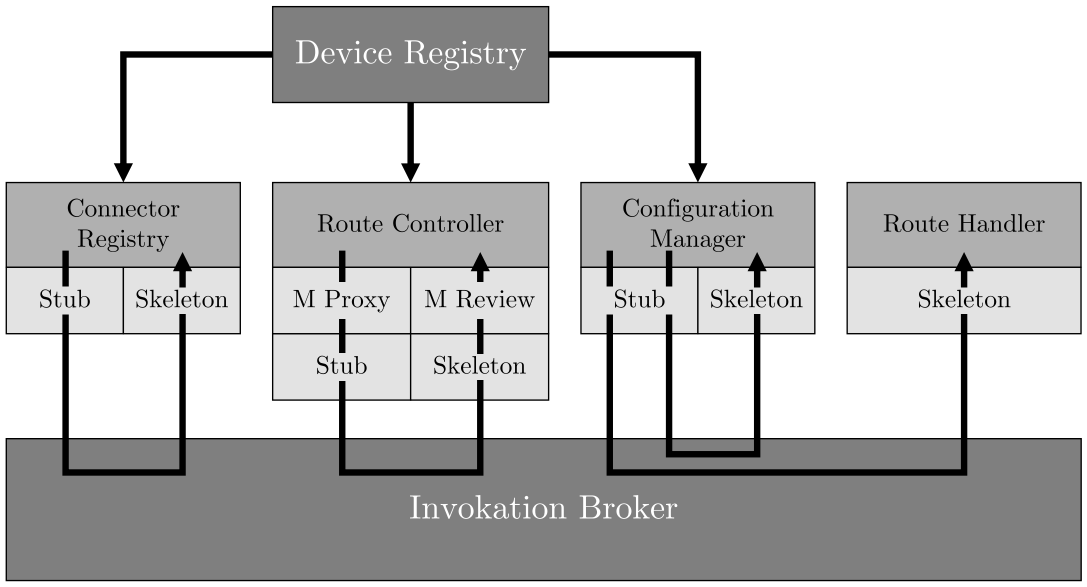


Figure 7.1.: The Connector Registry, Route Controller, Configuration Manager, and Route Handler utilize the BASE [16] skeleton and stub system for communication. Additionally, the BASE Device Registry is used to retrieve all available devices in the pervasive system.

BASE [16] is used as the underlying communication middleware for the prototype of the PerFlow Middleware. It offers the necessary functionalities for device handling and communication. The BASE middleware offers device and service discovery and allows to lookup all available devices in the pervasive system via its registries. Therefore, it is possible for PerFlow to obtain a global view of the system and to address each device with a unique ID assigned by BASE. Additionally, the middleware has a service based architecture, which allows for several services to run on each device. It is possible to let BASE automatically assign a random ID to each service or to define a fixed ID. If the ID is not known, it can be looked up in the service registry. The combination of device and service ID can be used to directly address each service in the pervasive system. Further, the BASE middleware offers Remote Procedure Call (RPC) [135] based invocations to communicate with offered services. The transmission of data in BASE is achieved by using a reliable TCP connection. The Connector Registry, Route Interpreter, and Configuration Manager are implemented as BASE services with a fixed ID. As they are executed on every device on startup, the PerFlow Middleware can communicate with them by only knowing the device ID due to the fixed service IDs. Additionally, the Message Proxy and Receiver discussed in Section 6.5.1 are utilizing the proxy and skeleton system of the BASE middleware. The PerFlow Middleware itself is responsible for serializing the information and figuring out the targets for a message using the Message Proxy as explained in Section 6.5.2. For sending the information, a BASE invocation is created for each target containing the sender ID, message type, data type, and the serialized data. In the next step, the invocation is handed over to the BASE middleware to deliver the message. To efficiently distribute information with a high number of targets, the serialization of data, creation, and sending of the invocation is parallelized. The abstract message receiver implemented by the application developers to receive information is created as a BASE service. Upon receiving an invocation, the message receiver is called by BASE and the PerFlow Middleware is again responsible for the deserialization of the incoming data. Figure 7.1 shows all the PerFlow classes directly connected to the BASE middleware and how the communication is achieved.

7.2. Implementation of the PerFlow Tool

In the following section, the implementation of the prototype for the PerFlow Tool is described in detail. First, the tool itself, the implementation of the UI, and the platforms used for the prototype is discussed. Second, the connection to the PerFlow Middleware is outlined. This section concentrates on the prototype of the tool itself, the visual scripting language is implemented as described in Section 6.6.1.

7.2.1. The Visual Scripting Tool

The PerFlow Tool prototype is implemented as an application for the Microsoft Universal Windows Platform (UWP) [122]. Therefore, it was implemented using C#. With the use of UWP the requirement R_{F4} for heterogeneity is supported.

While limiting the possible operating systems to Microsoft Windows, it allows for the execution of the tool on smartphones, tablets, and personal computers. Additionally, UWP also supports specialized devices like the Microsoft Surface Hub[1] without having to port the application to the new hardware. This is especially helpful in scenarios like smart classrooms or meeting rooms, which the PerFlow system should support as a possible use case. In these environments Surface Hubs are already starting to show up in a larger number and are used as smart whiteboards, which are also able to run UWP applications natively. Thus, the PerFlow Tool can be installed directly in the smart environments and, e.g., lecturers are able to reconfigure the middleware without the need of an additional device. The finished application can even be uploaded to the Microsoft App Store[2] for easy deployment on supported devices.

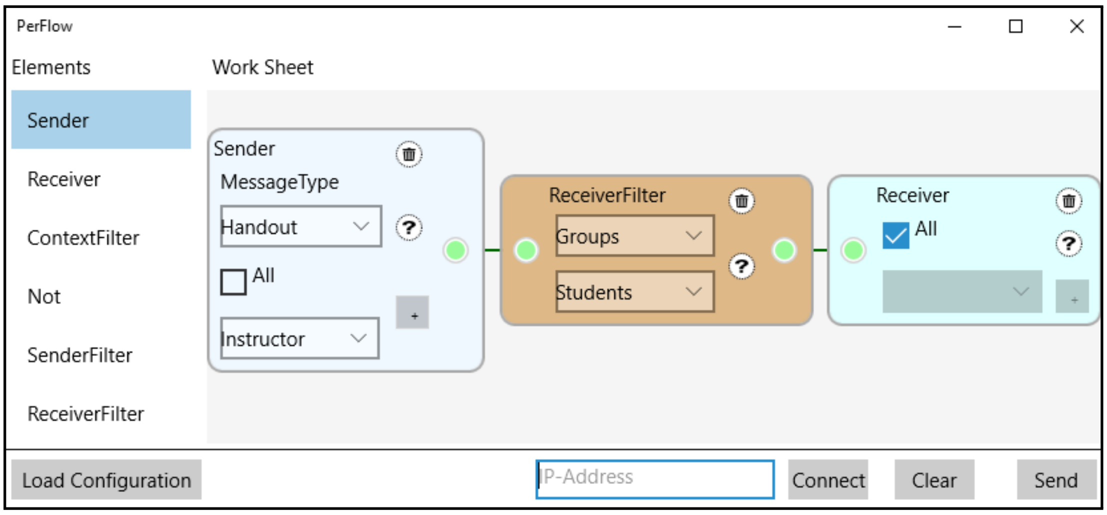


Figure 7.2.: The user interface for the PerFlow Tool showing the list of visual scripting elements on the left, the menu on the bottom, and in the middle the canvas containing a rule.

In Figure 7.2 a screenshot of the PerFlow Tool prototype is shown. The actual implementation is as close as possible to the design described in Section 6.6.2. One of the main goals was to provide a clean and understandable user interface to achieve the requirement R_{NF4}. Therefore, we considered different UI guidelines for the implementation of the PerFlow Tool prototype, e.g., from Apple [9], Microsoft [123], or Google [60]. The main considerations were to offer an

[1]https://www.microsoft.com/en-us/surface/business/surface-hub-2
[2]https://www.microsoft.com/en-us/store/apps/windows?source=lp

intuitive to use application which is focused to the key tasks it has to fulfill. Users should be able to learn the usage of the application fast and grasp the offered functionalities at the first glance. Nonetheless, the visual scripting tool should be functional and support a fast and effective work flow. The visual scripting elements are color-coded to assist the users at reading the rules and allow them to distinguish the elements fast. Sender nodes are colored in a light blue, receiver in a light green, and all filters in an orange tone. Additionally, correctly connected ports are colored in a bright green, while unconnected ports are gray. Further, the users interact with the tool via drag and drop for the creation of rule. Elements can be dragged from the list on the left side and dropped on the main canvas. Repositioning them is done in the same way. This type of interaction is especially useful on touch enabled devices like smartphones, tablets, or the Surface Hub. Figure 7.3 shows the example of a lecture with several students and a lecturer. Here, the lecturer is able to configure the information flow on a Surface Hub and distribute the presentation to the projector at the front and the laptops of the students.

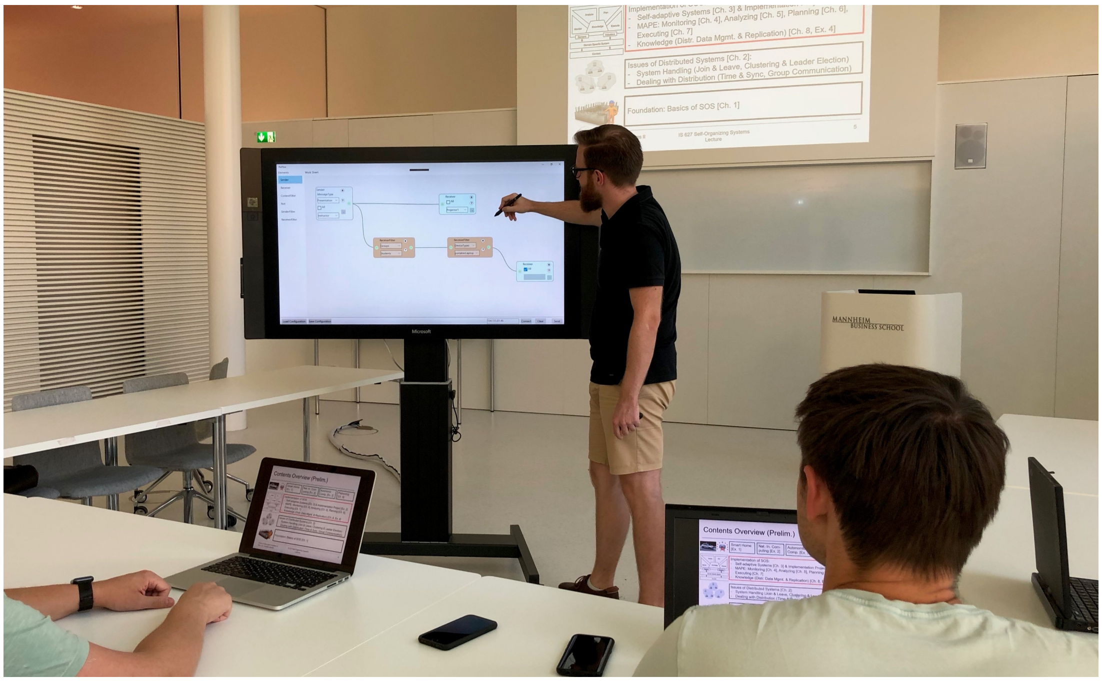


Figure 7.3.: An example setup of the PerFlow system in a smart classroom setup with one user creating new rules with the PerFlow Tool while the others see the current presentation on their device and the projector.

7.2.2. Communication with PerFlow Middleware

The PerFlow Tool and PerFlow Middleware are developed as separate entities, which in the future allows for an easy exchange with new visual scripting tools or other pervasive middlewares. As it is necessary for the tool to receive information about the current status of the pervasive system and for the middleware to get updated with the configuration created by the user, both need to be connected and communicate with each other.

To share the needed information, the PerFlow Tool connects to a Java proxy communicating with the PerFlow Middleware. As soon as the tool starts up it automatically searches on the local device for the proxy and connects to it. The proxy directly retrieves the current configuration and the information about all available devices from the middleware and forwards it to the visual scripting tool. Additionally, it is responsible for receiving the newly created configuration as XML, interpreting it, and handing over the rules as JSON to the middleware, as described in Section 6.6.3. For parsing the XML configuration of the visual scripting tool the Dom4j library [45] is used. The visual scripting elements and connectors contained in the configuration are then handed over to the JGraphT library [96], which generates a directed graph. In the next step, the Dijkstra based shortest path algorithm offered by JGraphT is used to generate the rules.

For the actual transfer of information between the tool and middleware the MQTT protocol [143] is used. MQTT is available both for Java and C# and thus can be used with the PerFlow Middleware and UWP. The proxy runs a MQTT publish subscribe broker, which the visual scripting tool connects to. Therefore, the PerFlow Tool is decoupled from the middleware and allows in the future for an easy exchange of the tool. While it is currently only available as an UWP application for devices running Microsoft Windows 10, it is possible to develop and use applications for other platforms, like Apple iOS or Google Android, without changing the PerFlow Middleware. As kibg as they are MQTT-enabled, it would even be possible to combine completely different approaches with the PerFlow Middleware, like a database of configurations where the user can pick a wanted set of rules.

7.3. Implementation of the PerFlow Virtual Extension

Similar to the PerFlow Tool, the PerFlow Virtual Extension is designed in two major parts. First, the actual virtual environment utilizing an existing game engine. Second, the Virtual Controller located at the pervasive middleware and providing the connection between the virtual environment and the PerFlow Middleware.

Therefore, the following section will first take a deeper look on the virtual environment and discuss the used game engine and libraries, and the developed prototype of a virtual classroom. Afterwards, the connection to the pervasive middleware and the communication between the physical and virtual environment is discussed.

7.3.1. Virtual Classroom Environment

The design of the PerFlow Virtual Extension is independent of any specific Game Engine but for the implementation of the prototype one has to be chosen. While the developers of early 3D games and environments relied on their own rendering methods developed in-house an increasing interest in third party game engines can be observed in recent years. Especially small to medium sized developers, as they are common in the field of educational/serious games and virtual environments, sacrifice the flexibility of implementing their own engine for the easier and time saving development with an out of the shelf game engine.

There are many commonly used engines with the most used ones being the Unreal Engine 4[3], CryEngine[4], Blender Game Engine[5], and Unity3D[6]. Out of the game engines presented in [156], we chose *Unity3D*. The CryEngine and Blender Game Engine are lacking behind in their development compared to their large competitors and the Unreal Engine is more focused on the highly professional market and large development projects. In comparison, the Unity3D engine also offers a high flexibility regarding devices, operating system (OS), and programming languages. It is possible to run virtual environments developed with Unity3D on

[3]`https://www.unrealengine.com/en-US/`
[4]`https://www.cryengine.com/`
[5]`https://www.blender.org/`
[6]`https://unity.com/`

all major OS including Microsoft Windows, Linux and Apple MacOS. Further, the engine also provides support for gaming consoles, like the Sony Playstation 4, Microsoft Xbox One, or Nintendo Switch, and mobile devices including Android and iOS. Additionally, Unity3D supports VR devices, like the Oculus Rift and HTC Vive, directly without the need to port the developed virtual environment. Such VR devices help to increase the immersion of environments developed with PerFlow Virtual Extension e.g., virtual smart classrooms. To aid the developer, Unity3D also provides them with further tools and libraries aside the 3D rendering engine. This includes a graphical world builder, animation tools, or libraries for artificial intelligence, sound, and networking.

Unity3D offers several programming languages to the developers including JavaScript, Boo, and C#. For the development of the PerFlow Virtual Extension C# was chosen, as it appears to have the highest support in the Unity3D developer community with a large amount of available documentation and tutorials. This increases the potential target audience for the PerFlow Virtual Extension. When developing with the Unity3D engine, everything included in the virtual environment is represented as a game object. Each of these objects can be provided with several properties defining the behavior or visual appearance of it e.g., textures, collision meshes, or scripts programmed by the developer containing the game logic. These game objects could be virtual objects (e.g, chairs or tables), terrain and buildings, or avatars. Additionally, it is possible to create empty game objects, which are present in the environment during runtime, but have no visual representation. This is useful for adding logic and services to the virtual environment which are universal and not affiliated with any visual objects like the player avatar. The complete implementation of the Virtual Broker for the PerFlow Virtual Extension was developed as C# scripts and added to such an empty game object. Developers are provided with a preset of this object, which they can just drag and drop into their virtual environment to add support for PerFlow.

For the prototype of the PerFlow Virtual Extension an example virtual environment was developed, which allows for testing, debugging, and evaluating the system. Fitting to the scenario discussed in Section 4.1 a virtual classroom was created, which can be seen in Figure 7.4. It is the representation of a physical classroom and contains a main projector for presentations, a public display at the

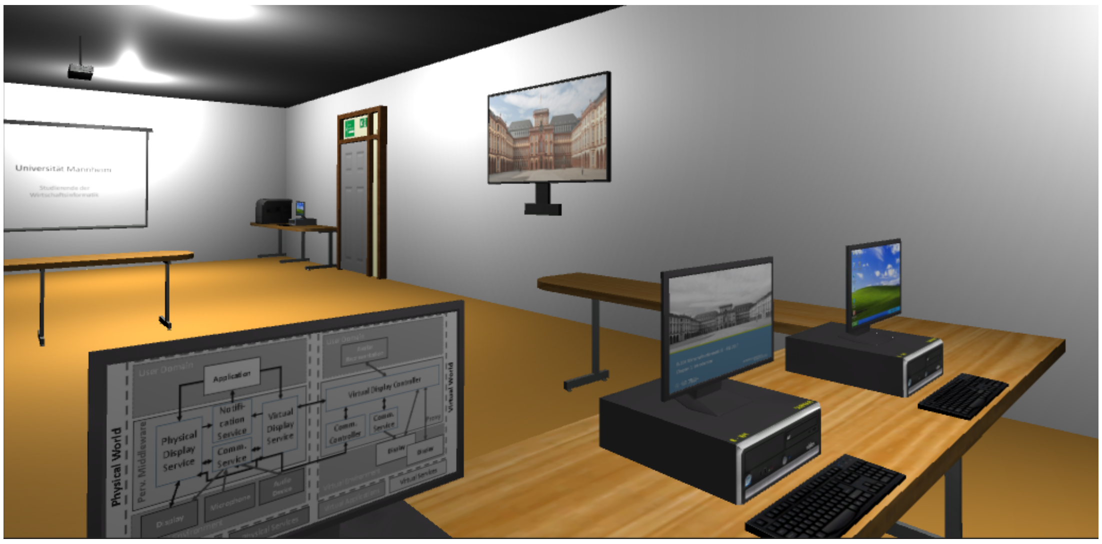


Figure 7.4.: A virtual classroom with several screens using the PerFlow Virtual Extension to receive content like pictures or presentations. Additionally, the user has the ability to communicate via the middleware with text or speech chat.

side, and several PCs with screens at each seat for the students. All these screens are connected to the Virtual Broker and are able to receive and show images. Additionally, the virtual environment contains avatars representing the user and other students or lecturers in the system. These avatars are also communicating via the Virtual Broker and are able to send and receive text based chat messages and also voice chat.

7.3.2. Communication with PerFlow Middleware

The PerFlow Virtual Extension is communicating with the middleware via proxies similarly to the PerFlow Tool. As described in Section 6.7, a proxy is started for each virtual application running in the PerFlow Virtual Extension and connects it to the PerFlow Middleware. For the communication between the Virtual Controller and Broker they are connected via a reliable TCP connection and the previously defined protocol is implemented. They are also responsible for the serialization of the data transmitted between them. As the controller is implemented in Java and the broker in C#, *Protocol Buffers* [62] is used as an external serialization library. It offers a fast, convenient, and platform-independent serialization [53]. As ProtocolBuffer does not offer a stable C# library, we also use the *protobuf-net* library [157] for serialization within Unity3D. For the actual transmission the standard implementation of the TCP sockets is used on the Java side and the *UNET* package provided by Unity3D is used by the Broker.

Regarding the data handling, Unity3D offers direct support for the image formats Joint Photographic Experts Group (JPEG) and Portable Network Graphics (PNG) and the video format Ogg. The engine is able to render the content of these files directly in the virtual environment without the need for further libraries. Presentations in the Portable Document Format (PDF) file format are split up by the Virtual Controller into single PNG images containing each slide before sending them to the virtual environment. Therefore, no PDF support is needed in the Unity3D engine. For splitting up the presentations the GhostScript library [10] is used. While these formats cover the implemented display service, the chat service needs additional data types. For the text based chat simple strings are used and transmitted via the PerFlow Middleware to other participants. To enable the voice chat, the additional library *Nspeex* [141] for recording the sound and serializing it into byte arrays is used. It gives access to the microphone connected to the user device and is able to capture the audio signal. The recording frequency is set to 16,000 Hertz. In the next step the signal is split into chunks and decoded. These individual chunks are then transmitted as a byte array and on the receiver side deserialized using the same library.

7.4. Summary

During this chapter we presented the prototype implementation of the design for the PerFlow system. Therefore, we discussed the PerFlow Middleware, PerFlow Tool, and PerFlow Virtual Extension separately. In the prototype we used BASE [16] as a communication middleware, UWP [122] as the platform for the visual scripting tool, and Unity 3D [156] as the game engine for the virtual environment. The prototype covers the complete design discussed in the previous chapter and is the foundation for all following conducted evaluations.

8. Evaluation

After discussing the complete design of the PerFlow system the previous chapter introduced the prototypes for the PerFlow Middleware, PerFlow Tool, and PerFlow Virtual Extension. Thus, the next chapter concentrates on the evaluation of these prototypes. First, a proof of concept shows the feasibility of the system by introducing an example smart classroom. Second, the effort a developer has, to use PerFlow with his application is showcased in Section 8.2. Third, the performance of the implemented prototypes is measured. Finally, Section 8.4 introduces a user study to evaluate the perceived ease of use and usefulness of the PerFlow Tool.

8.1. Proof of Concept

To show the feasibility of the proposed middleware and how it can be used in a real environment, a smart classroom was implemented as the chosen use case. The proof of concept for the PerFlow system is based on the testbed for pervasive middlewares in learning environments (PerLE) introduced in [134][1]. Following, the concept and services of PerLE are introduced before evaluating the requirements with the help of the testbed. While the in [134] introduced PerLE testbed offers a total of five different services for the use in a smart classroom, it was later extended with a sixth live feedback service. The proof of concept will concentrate on three of them and how they utilize the PerFlow Middleware.

8.1.1. The PerLE Testbed for Pervasive Classrooms

PerLE offers lecturers and students an application to use during lectures combining six different services. The main goal is to increase the efficiency of the

[1] [134] is joint work with S. Schmitz and C. Becker

lessons by incorporating all available devices and enable them to share information for a better transfer of knowledge. Following, the overall concept of PerLE is discussed together with the use case and the applications handed out to the users. Afterwards, at the example of three of the services, the migration to the PerFlow Middleware is discussed and how its functionality is utilized for the testbed.

Concept

In today's learning environments, e.g., classrooms or lecture halls, an increasing amount of devices brought by students, lecturers or belonging to the infrastructure are available. The goal of PerLE is to offer the users in such environments different services to increase the learning experience. Additionally, an application is provided for students and lecturers to install on their devices. The prototype of PerLE is implemented in Java and uses the BASE middleware [16], however the design allows for the middleware to be exchanged with little effort. Thus, for the proof of concept it was switched to the PerFlow Middleware. The testbed allows for performance measurements and shows how the evaluated middleware works in a real world scenario.

The application for students and lecturers bundles the services included in PerLE. Thus, it offers functionalities like access to projectors or displays, file transfer, surveys, or live feedback during the lecture. The application can cope with a high variety of different use cases which may occur in ha pervasive classroom. These scenarios could be a standard head on lesson, group work, or presentations by different students. Especially together with the PerFlow Tool, the lecturer is able to react to these changing use cases and reconfigure the pervasive system on the fly. Figure 8.1 shows an example lecture, where one person is holding a presentation while the listeners are able to follow the slides on their personal devices, download them, and give feedback.

Services

The PerLE testbed offers a total of six different services for the use in pervasive classrooms. A *display service* can be used to share projectors and screens in the

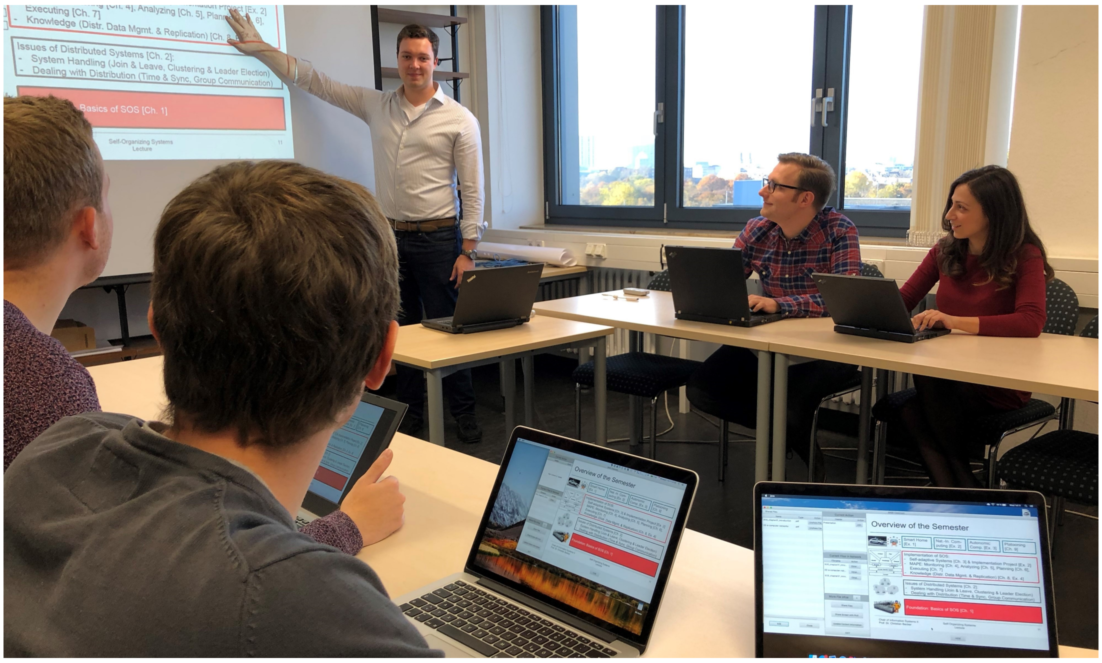


Figure 8.1.: The PerLE system deployed in a real lecture room. While only the presentation is shown on the screen, all other devices show the complete application with access to all services contained in the testbed.

lecture room and show content such as presentations. The *file service* allows for the peer-to-peer distribution of files between applications. With the *survey service*, the lecturer can distribute predefined surveys and tests to students and collect the results. A *user service* offers the lecturer the possibility to collect the contact data of the students present in the lecture. Lastly, the *room control service* gives access to smart devices in the system like shutters or the lighting. In addition to these five services introduced in [134], a *live feedback service* was developed, which, in contrast to the *survey service*, allows for continuous feedback from the student to the lecturer e.g., regarding the speed, volume, or questions on the content. Following, the *display*, *room control*, and *survey service* are discussed in more detail and the integration of the PerFlow Middleware is explained. The applications for the lecturer and students register with the Connector Registry of the PerFlow Middleware. They provide properties for the device type (e.g., PC, tablet, or projector), the role (lecturer, student, or infrastructure), and location (remote or local).

Display Service: This service is designed to be used in two different ways. First, it can be executed as a standalone service directly on devices like projectors or public displays in the infrastructure of the pervasive classroom. This is similar to the initial service introduced in [133][2]. Second, it is also integrated into the application used by the lecturer and students. Thus, the content can not only be displayed at the front of the classroom but also directly on the users' devices. The display service supports images and PDF files for which the PerFlow Middleware is extended with a new custom object for transferring them. If a presentation in the PDF format is received, the GhostScript library [10] is used to decompose it into single images for each slide to be able to display them. The display service registers connectors at the PerFlow Middleware for receiving images and PDF files, and a connector to send commands (as strings), to forward or reverse the presentation. These connectors are bundled, as while displaying a presentation it should always be possible to control it. The application for the lecturer and students has registered the corresponding outgoing connectors. Additionally, the display service is available remotely through the PerFlow Virtual Extension, where the handling of the incoming images and PDF is done similar to the approach described in [131][3].

Room Control Service: The room control service acts as an intermediate between the PerFlow Middleware and typical smart devices from the home automation market, like smart light bulbs or automatic window shutters. It therefore makes use of the Representational State Transfer (REST) API provided by many of these smart devices, e.g., as by Nest[4]. It would be possible to extend the service in the future if additional APIs need to be supported. For the prototype, the iCasa simulator developed by Lalanda *et al.* [104] is integrated. The service offers a list of available smart devices and their capabilities as a custom transfer object. Additionally, it can receive strings with commands which are then forwarded to the smart device. For the device list and the commands it registers the corresponding connectors, which are also bundled. The application provided to the users is able to receive the list and display the content, so that

[2] [133] is joint work with D. Schäfer, S. VanSyckel, and C. Becker
[3] [131] is joint work with C. Krupitzer and C. Becker
[4] `https://developers.nest.com/guides/api/rest-guide`

the user can interact with the smart devices and e.g., send the command to turn on the lights.

Survey Service: The third service discussed in this section allows the lecturer to send out surveys to students. For instance, lecturers are able to collect feedback after the lesson. These surveys are predefined in the JSON format. The lecturer only needs to choose a survey, which is afterwards distributed to all student devices. The receiving applications then interpret this survey and present the students the corresponding UI elements (e.g., radio buttons or text boxes). After filling out the survey, the responses are then sent back to the lecturer, where they are collected. For the transfer, the application of the lecturer is registering an outgoing connector for the surveys and an ingoing for the results. The student applications are registering their connectors vice versa with the Connector Registry of the PerFlow Middleware.

8.1.2. Requirements Evaluation

The previous section introduced the PerLE testbed with its functionalities, the following section performs a qualitative evaluation. Therefore, the application and services included in the testbed are used to demonstrate the fulfillment of the requirements introduced in Chapter 4.

The PerFlow Middleware allows for the information exchange between the applications of lecturers and students. Also, it is possible to communicate with infrastructure devices like projectors via the display service or light bulbs and shutters via the room control service. This supports requirement R_{F1}. To achieve this, PerFlow uses BASE as a communication middleware. As shown at the three presented services for the testbed, developers have the possibility to bundle the information flow for their applications and services (R_{F3}). Additionally, they are able to introduce new content types for the transfer via the PerFlow Middleware e.g., the PDF files for the display service, which supports the requirement for extensibility (R_{NF5}). With the help of the PerFlow Virtual Extension, it is also possible to migrate the services from the physical into the virtual environment. Thus, the display service is able to present the same content to both the local and remote students. This enables a remote access to the system, which is as close as possible to the experience in the smart classroom itself (R_{F7}). The lecturer in the introduced PerLE testbed is able to reconfigure the information flow in the pervasive system with the help of the PerFlow Tool supporting requirement R_{F2}. This could include giving specific students access to the projector or specify which device should collect the responses for a survey. As in a classroom not everyone should be able to influence the complete system to prevent chaos, the possibility to reconfigure the PerFlow Middleware should be left only to the lecturer. Therefore, the Consensus Module introduced in Section 6.4 allows for an access control and for limiting the right to send new configurations to the PerFlow Middleware to specific users or groups (R_{F6}).

The requirement R_{F4} for the support of a heterogeneous system is fulfilled in two different ways by PerFlow. For the possibility to include a high range of different devices and offer information about the devices available in the system (R_{F5}), a communication middleware is used. In the prototype, BASE [16] provides the needed functionalities to fulfill this requirement. Additionally, the PerFlow Middleware should not only connect a large amount of devices but the tools and systems introduced with PerFlow also need to provide a possible large interoperability. By using UWP for the PerFlow Tool and the Unity3D game engine for the PerFlow Virtual Extension it is possible to execute them on many different devices. This includes smart boards like Microsoft Surface Hub for reconfiguring the pervasive system or VR devices like the Oculus Rift for participating remotely. While the PerLE testbed is targeted to smart classrooms, the same services and mechanisms could be used in several other use cases e.g., smart meeting rooms or conference centers. Further, in the user study in Section 8.4, an additional use case for a smart airport lounge is introduced. Thus, it is shown that the PerFlow system is flexible and can be tailored towards a high number of different scenarios supporting the requirement for generalizability (R_{NF3}).

8.2. Implementation Effort

To ensure a widespread adoption, using the PerFlow Middleware should lead to minimal effort for the developer. Thus, it is essential that the overhead for the developer is as low as possible to encourage a widespread adoption.

The implementation effort is compared to using the BASE middleware for the communication between applications and services without PerFlow. To quantify the implementation effort with and without PerFlow Middleware, the concept of source Lines of Code (SLoC) is used [6]. The SLoC metric is further differentiated into physical and logic SLoC. While counting the lines of code physical SLoC excludes comments and empty lines from the measurement and logical SLoC additionally takes different ways of writing the same code into account e.g., curled brackets at the end of the line or in a new one [138]. For the measurement the Statistic[5] plugin for the IntelliJ[6] Integrated Development Environment (IDE) is used.

The examples offer a simple way for users to share the content of their screen with others. Therefore, an application sending screenshots and a service receiving the screenshots was implemented for each BASE and PerFlow Middleware. The applications capture the screen of the source device in regular intervals (every two seconds) and send the resulting image with either BASE or the PerFlow Middleware to the target device. The services wait for incoming images and show the to the user with an image viewer. An excerpt of the source code for the most relevant part of the applications and services, with BASE and PerFlow, is shown in Appendix A. These excerpts larger helper functions for e.g., setting up and starting the middlewares (*setup(...)*), capturing the current content of the screen as an image (*ScreenCapture.capture()*), and displaying the incoming image (*viewer.showImage(...)*). The application using the PerFlow Middleware starts by registering an outgoing connector for the captured image and creates the message proxy. Afterwards, it hands over a new image every two seconds to the proxy and the middleware is responsible for distributing the image according to the current routes. In the case of the application developed with the BASE middleware, the setup and sending of information is more complex. While developing the application, the service and its interface used during runtime has to be known by the developer. During runtime the application needs to search for available services on its own, decide which services to use, and send out the image to them.

The logical SLoC is 18% lower for the application using the PerFlow Middleware (49 to 58 lines), while at the same time offering more features to developers

[5]`https://plugins.jetbrains.com/plugin/4509-statistic`
[6]`https://www.jetbrains.com/idea/`

and users. The BASE application is using a naive approach sending out the image to every display service no matter if it should receive the screen capture or not. This could lead to security problems (sending your screen to strangers) or performance problems (sending to too many devices at once). If the developers would like to control the communication more fine grained, the possible receiving services have to be checked manually and the developer is responsible for storing and distributing information needed for the decision. Therefore, the advantage of the PerFlow Middleware regarding the implementation complexity would further increase. This section looked at the implementation of applications with both middlewares. When looking at services receiving the information, the advantage of the PerFlow Middleware even increases, as the effort to implement a receiving application is not higher as the already discussed sending application. At the same time the complexity for a BASE services rises, as the developer would also need to implement a skeleton and proxy for the service. Thus, the logical SLoC for the services is 110% higher for the BASE service over the receiver service implemented with PerFlow Middleware (37 to 78 lines). In conclusion, the requirement R_{NF4} for the usability of the middleware is fulfilled for the developers, as PerFlow is able to ease their effort for implementing new applications.

8.3. Performance Measurements

Responsiveness is a major non-functional requirement of the system (R_{NF1}). As PerFlow is tailored towards everyday environments e.g., smart classrooms or meeting rooms, the users should not experience any hindrance due to the middleware. There are three possible functionalities which introduce an additional overhead compared to a traditional pervasive middleware. First, the reconfiguration of the system where the current status has to be collected and the new configuration needs to be distributed. Second, while sending information over the middleware, the receiver is not directly specified but has to be determined by the PerFlow Middleware. Third, if users take part in the system remotely, the information not only needs to be transmitted by the middleware but also has to be handed over to the PerFlow Virtual Extension.

In the following sections performance measurements are conducted for these three processes to evaluate the overhead introduced by PerFlow. The computers used for these evaluations are two PCs with an Intel Core i7 8700k hexacore CPU with 3,70GHz, 32 gigabytes of memory, and a Nvidia Geforce GTX 1080Ti graphics card running Windows 10 64bit and Java version 10.01.

8.3.1. Reconfiguration Overhead

Reconfiguring the information flow in the PerFlow system triggers several events and the process has to be coordinated between all available peers in the system. The reconfiguration can be started in two different ways: By a change in the pervasive system or by a user introducing a new configuration. Following, the performance of the second case is evaluated in more detail, as it is the more complex process.

The reconfiguration starts, when the user, e.g. the lecturer, sends a new configuration to the middleware. During this step, the available connectors are polled from all devices, the rules are interpreted, and the new configuration, as well as the new routes are delivered to all devices in the system. As there is network traffic involved in this process, the evaluation was conducted on two PCs connected in a private network via a Gigabit Switch. The first computer ran the application, which introduced the new configuration, while the second computer simulated up to 100 devices with one application having ten connectors each. Thus, the first computer received the available connectors and had to send out the configuration and routes via the network. Two different JSON configurations were used for the performance measurements, a small one containing 5 and a large one with 20 rules. As the user would have to create these configurations during runtime using the PerFlow Tool a total of 20 rules is already a large but still feasible size for the configuration. For the increasing number of devices and the two configurations each single measurement was performed 100 times to create an average and mitigate the influence of outliers.

Figure 8.2 shows the results of the evaluation with the average time needed to reconfigure the system with an increasing amount of devices. It includes also both configurations, with 5 and 20 rules. The results separated by the size of the JSON configuration can be seen in Appendix B. On average, the reconfiguration of the middleware needed 57.08ms, where the majority of the time was needed to distribute the configuration to all devices (32.01ms) via the network. This

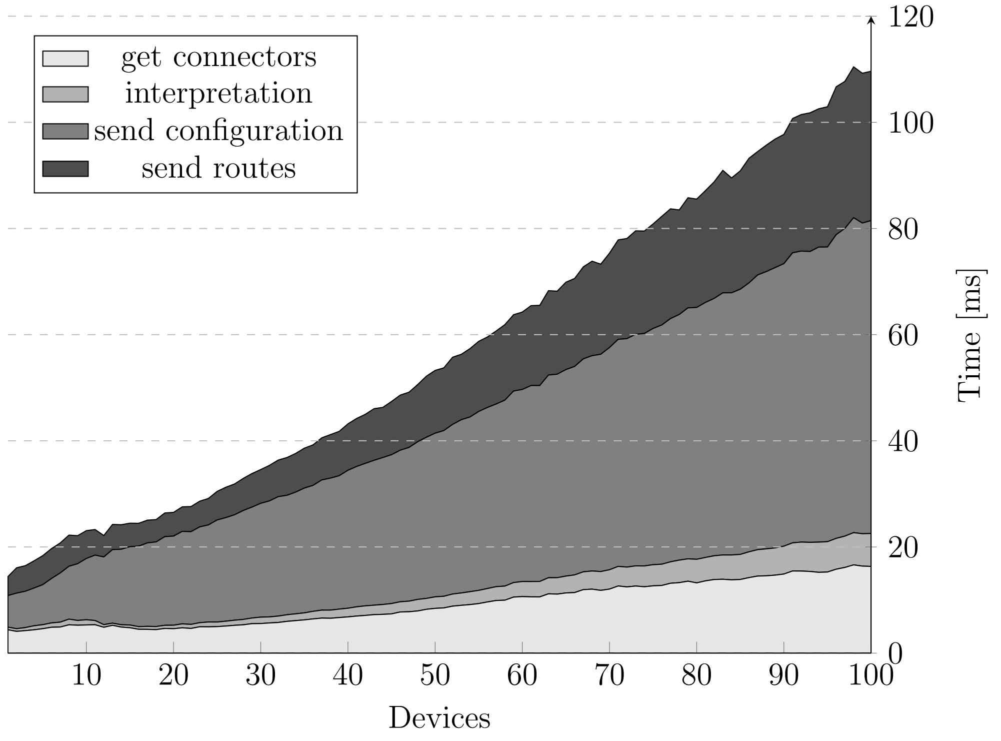


Figure 8.2.: Time needed to reconfigure the information flow for up to 100 devices. The values are measured with two configurations containing 5 and 20 rules and the average times are used for this chart.

distributed configuration includes the JSON rules for the PerFlow Middleware and the XML representation of the visual scripting elements for the PerFlow Tool. The average for the 5 rule configuration was 45.59ms and for 20 rules 68.58ms. For the maximum of 100 active devices we see an average time of 87.78ms for 5 rules and 132.47ms for 20 rules. Thus, even with 20 rules, which would already be complex to specify using visual scripting, the reconfiguration time would not lead to noticeable delay in the workflow of the users. An uninterrupted workflow is also ensured as the previous configuration is still active and usable until the reconfiguration is finished. The effort of reconfiguration rises linearly with the number of devices present in the system. While the longest part of the process is the delivery of the configuration to all devices, the actual interpretation takes only 2.59ms on average.

In the case of a change in the pervasive system without the introduction of new rules, many of the steps described above still apply. The connectors of the available devices need to be collected, the existing rules are interpreted, and the new routes are distributed. As the configuration itself did not change, the main advantage is that it does not need to be distributed. Thus, the process is sped up drastically, as the most time consuming part can be ignored.

8.3.2. Consensus Algorithm Overhead

In the previous section, the time needed to perform the reconfiguration for the information flow was discussed. This applies in the naive approach, where everyone is allowed to introduce new configurations to the system without any access control. If the administrator wants to only give some specific users or user groups the right to reconfigure the system, it is possible to set the PerFlow Middleware up accordingly as discussed in Section 6.4. While the process and effort of reconfiguring the system stays the same, additional steps are introduced to coordinate the access control, which may have an influence on the performance. According to the consensus algorithm explained in the design, the system needs to elect a leader which is then responsible for deciding if the requesting peer is allowed to apply a new configuration. In a first step, the effort for the election of a new leader is measured before discussing the overhead for evaluating the permissions.

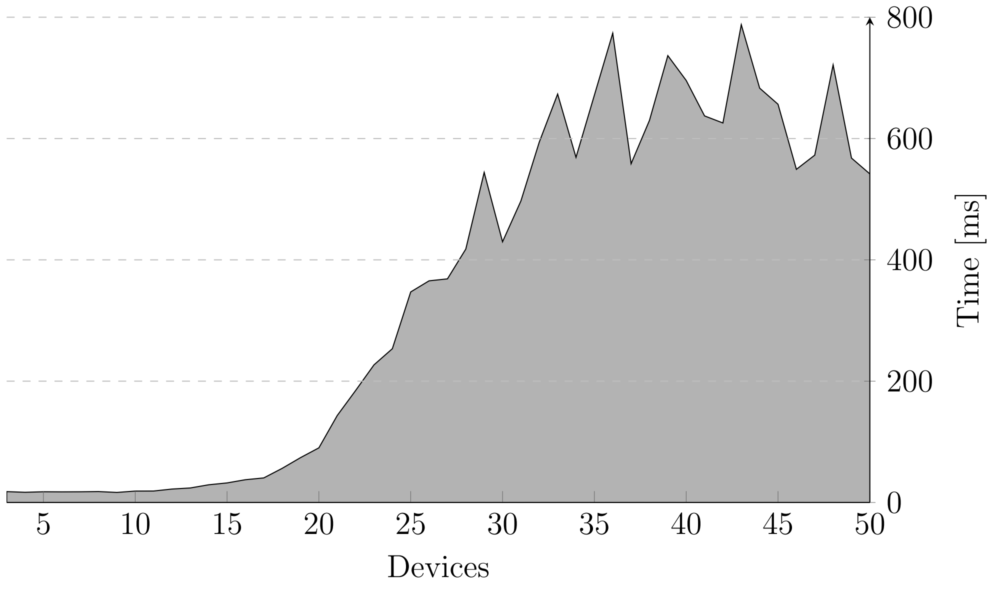


Figure 8.3.: The average time (in ms) needed to elect a new leader for the consensus module measured for growing pervasive systems of up to 50 devices.

While the election process is quite complex and requires a lot of coordination between the different peers in the pervasive system, it also should not be needed often. In the best case scenario the system is very stable by e.g., electing the device of a lecturer as the leader which possibly will not leave the system during the complete lecture. But even in more fluctuating scenarios the new elections should not interfere with the user experience. For the evaluation the time needed for the complete election process was measured. The time was measured from the point, where the old leader is shutdown and the election algorithm is triggered until the process returned with a new leader or in the worst case failed due to a time out. The evaluation was executed with an increasing amount of connected peers until a total of 50 devices was reached. The results for the measurements can be seen in Figure 8.3. For an increasing number of devices in the pervasive system the time needed to perform the election first increases rapidly until settling between about 600ms and 800ms after reaching a total of 32 devices. This effect has two different causes. First, the load on the PC used for the evaluation is increasing heavily leading to slightly longer response times. But the second and more relevant cause is, that the number of devices recognizing the loss of the leader rises and therefore, increasingly more messages are generated announcing the need for a new leader and voting on the possible new leaders. The reason why the overall time needed does not rise further, is the timeout of 800ms for the election process. If the peer responsible for coordinating the election is reaching this timeout, it will ignore further messages and determine the winner of the election. The prerequisite therefore is, that over half of the available peers actually voted. If this is not the case the election failed. Thus, a fitting timeout for the expected size of the pervasive system is important, as a too large value would lead to longer and a too small value to more failed elections. This is in the responsibility of the administrator deploying the system. For the evaluation the chosen timeouts did not lead to any failed elections.

The second overhead introduced by the consensus algorithm is for evaluating the permissions before sending out the new configuration. This is done locally on the elected leader and if the evaluation turns out positive, the transmission is similar to the process discussed in Section 8.3.1. The measurement for the permission check was conducted with an increasing number of devices in the system up to a total of 50 devices. It started when the leader received an inquiry to distribute

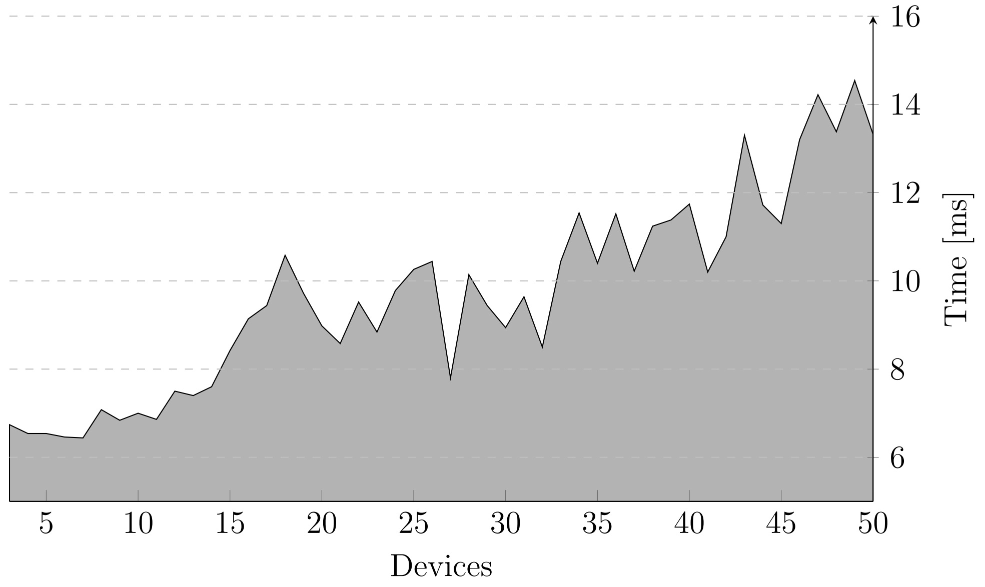


Figure 8.4.: The average overhead introduced by the permission evaluation executed by the leader upon receiving a new configuration.

a new configuration and ended as soon as he was finished with the permission evaluation and the distribution process started. The results of this overhead evaluation can be seen in Figure 8.4, which also shows the time needed on top of the previously discussed configuration distribution if the consensus algorithm is enabled. Overall, the computation time increases linearly with the number of devices, but even in the worst case it barely exceeds 14ms. Thus, the introduced delay is not recognizable for the user and does not interrupt the workflow.

In summary, the major overhead introduced by the consensus algorithm is the election of new leaders. While this process needs a substantial amount of coordination, it has not a high impact on the usability. In scenarios like smart classrooms or meeting rooms it is normally possible to elect a stable leader and thus reduce the need of new elections. Additionally, reconfigurations of the complete pervasive system, and with that the introduction of new rules and the need for the leader, is not extremely frequent. Therefore, the worst case, that one device has to wait for publishing its new configuration until an election phase is finished, is not very likely. The actual impact on the normal operation of the PerFlow Middleware, when the leader is stable, is very low and does not influence the experience of the users.

8.3.3. PerFlow Middleware Communication Overhead

For each outgoing message the PerFlow Middleware first has to determine all possible receivers according to the routes. This extra step in the process means the introduction of an overhead. Following, the influence of this overhead on the PerFlow system is evaluated by measuring the time needed to send messages between two applications. As BASE is used as the underlying communication middleware for PerFlow, it is also used as the baseline for the evaluation of the communication overhead. Therefore, each measurement is done first only with the BASE middleware and second with the addition of the PerFlow Middleware. These measurements were conducted on a single PC, because the determination of possible receivers is only performed locally and not influenced by messages received from other peers over the network. Thus, a local setting was chosen to eliminate possible uncertainties introduced by network delays. As a payload for the measurements, two uncompressed pictures with 500x500 and 1000x1000 pixels were used, which resulted in approximately 0.75MB and 3.0MB of transmitted data to each receiver per message. The overhead was measured for 5, 20, and 50 receiving applications and repeated for 100 times per image. Each measurement was performed with and without PerFlow.

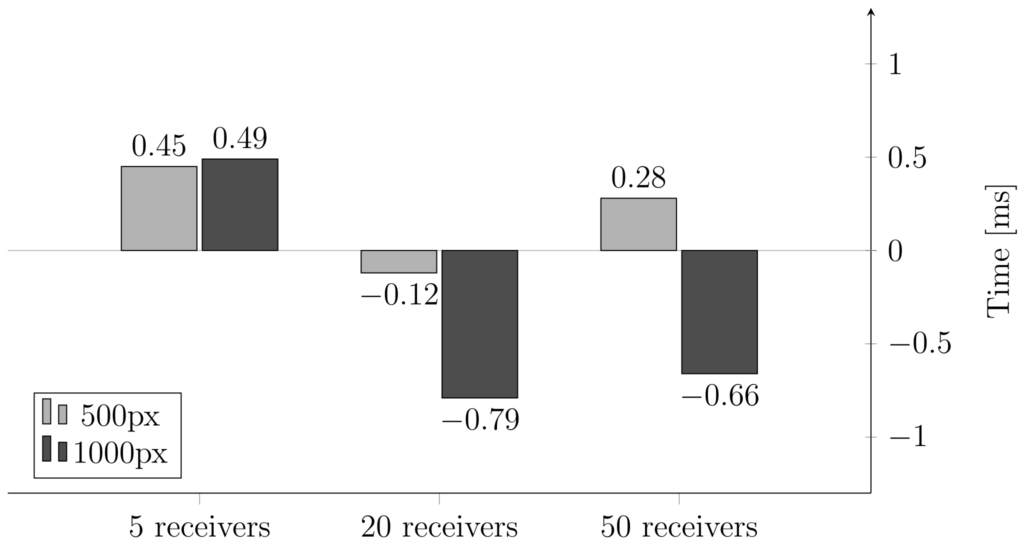


Figure 8.5.: Overhead introduced by the PerFlow Middleware. The values represent the delta to a communication via the BASE middleware only.

Figure 8.5 shows the overhead generated by the Route Controller of the PerFlow Middleware compared to a direct communication via the BASE middleware. Independent of the payload and the number of receiving applications, the difference in execution time is with values between +1ms and -1ms within the uncertainty of measurement. The interpretation of the rules contained in the JSON configuration is only executed once beforehand to generate the routes. Therefore, the main computational work is already done, when a message is transmitted and the determination of the receivers is only a mere lookup of the routes. This leads to the conclusion, that the PerFlow Middleware does not introduce a recognizable overhead during sending messages and is not affecting the responsiveness of applications.

Additionally, the PerFlow Middleware could even outperform the communication via BASE in some circumstances. This may be counterintuitive, as the BASE middleware is still used for the transmission of each single message. The effect can be credited to a better and more effective parallelization of the data serialization and the calls to the BASE middleware for sending the data. Thus, especially with higher amounts of receivers and larger payloads, it is possible for the PerFlow Middleware to be more effective while sending the data.

8.3.4. PerFlow Virtual Extension Communication Overhead

To evaluate the performance of the PerFlow Virtual Extension, we measured the time an image needs from being sent by an application until it is displayed at a virtual device. For the measurement we use three images as payloads with 1000x1000, 500x500, and 250x250 pixels. The images are transmitted uncompressed and are thus representing a payload of 0.19MB, 0.75MB, and 3.00MB. As Figure 7.4 shows, the virtual displays cover only a limited area of a physical display. Therefore, these resolutions of the images are sufficient for the use case. The transmission, and therefore the measurement, is split into two parts: first, sending the image from the application to the Virtual Controller and second, handing it over to the virtual environment and displaying the image. The first step contains the serialization of the image at the application, sending it via the PerFlow Middleware, and deserializing it at the Virtual Controller. In the second step the data is again serialized at the Virtual Controller, handed over to the virtual environment developed with the Unity3D engine, where the image is deserialized and displayed. The evaluation is repeated 300 times per image.

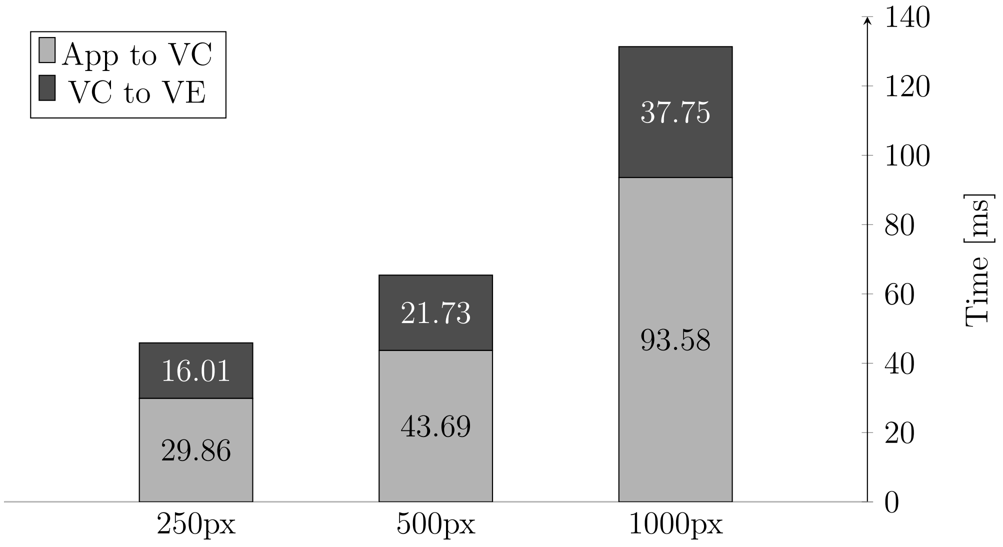


Figure 8.6.: The overhead introduced by the PerFlow Virtual Extension showing the time needed to forward the information to the correct virtual device. The measurement is split into sending from the application to the Virtual Controller (App to VC) and from the Virtual controller to the displays in the virtual environment (VC to VE).

Figure 8.6 shows the time needed in total and for each of the two steps. The measurement was conducted with one application sending to one virtual display. The effort needed to send the image to the Virtual Controller is similar to sending to a physical display. Thus, the main overhead of the PerFlow Virtual Extension is in handing the data over to the game engine and displaying it in the virtual environment. Even for the largest image, the time needed to send it from the Virtual Controller to the virtual display is with 37.75ms not recognizable for the user. If the virtual environment is executed with 30 frames per second, the measured delay is under two frames (one frame equals 33.33ms). This does not influence user experience as for users it seems as the physical and virtual displays are synchronized. For a single image the transmission via the PerFlow Middleware consumes up to 71% of the needed time from application to virtual display (69% on average).

Next, we evaluate how the PerFlow Virtual Extension scales with an increasing amount of virtual devices. Therefore, we measure the time needed

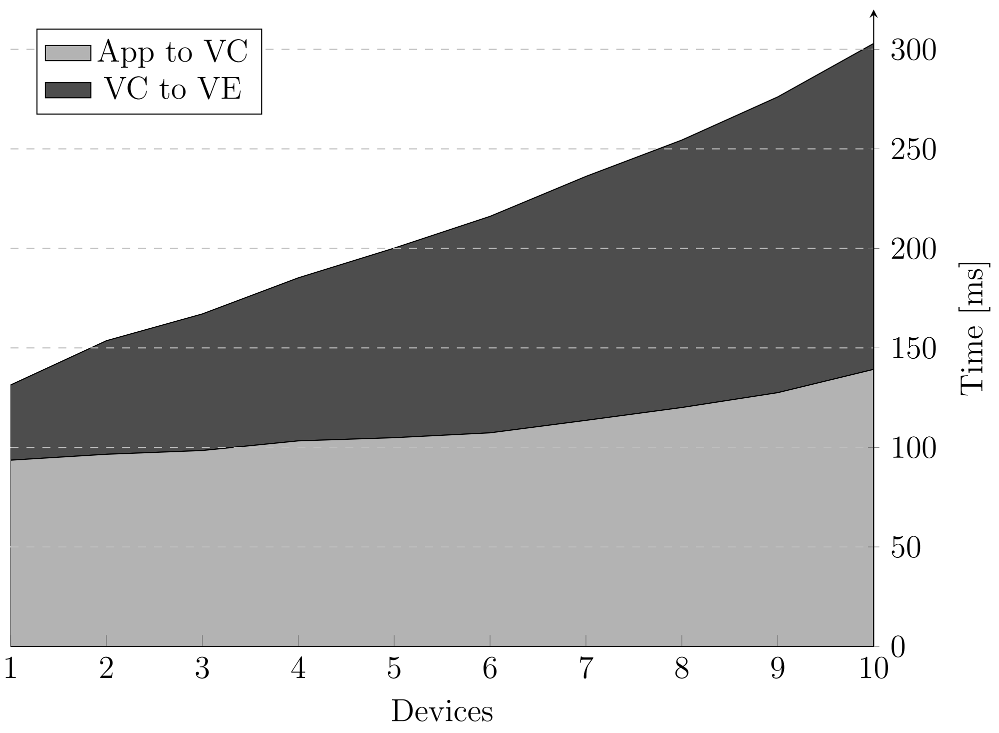


Figure 8.7.: The transaction time needed to send the same picture to an increasing number of services in the PerFlow Virtual Extension.

to send an image to several virtual displays. The evaluation again measures the time from the application to the Virtual Controller and from there to the virtual displays. For this measurement we used the largest image with 1000x1000 pixels. The amount of virtual displays was increased up to ten displays placed in the virtual classroom. Figure 8.7 shows the resulting transaction times. The average increase for the complete process is 19.06ms for each additional display. To send an image to all ten displays simultaneously, PerFlow needs on average 302.94ms. In an optimized scenario it is highly unlikely for all applications to update their virtual display at the same time. The time needed by the PerFlow Middleware to send the image to the Virtual Controller increases only by 5.07ms per additional display. Therefore, with only a few displays the effort to distribute the image in the pervasive system is larger then handing it over to the virtual environment and displaying it. Starting with six simultaneously receiving virtual displays, we see a shift in the effort needed and displaying the image gets more time consuming. Nonetheless, even in the worst case displaying the image takes 164.73ms, which accounts for less then 5 frames, when the virtual environment is executed with 30 frames per second. This is not noticeable by the user in the typical use cases of the PerFlow system, e.g., a presentation, where such a delay in a slide change would not be interruptive.

8.4. User Study

As the PerFlow Tool is mainly aimed at end users with little to no IT knowledge, one of the main requirements is a high usability (R_{NF4}). The user has to understand the tool fast and be able to create new rules for the pervasive systems without having a deeper knowledge about the theory behind it. To evaluate, if the users are able to really use all the offered functionalities of PerFlow and if they are able to do so without trouble, two user studies were conducted.

The following section first introduces the methodology of the user study and the overall procedure. Afterwards, the participants of the study and their recruitment are discussed before taking a deeper look on the tasks they had to perform and on the data acquisition. Lastly, the results of the study are shown and the assessment of the PerFlow Tool by the users is discussed in detail.

8.4.1. Methodology

For the evaluation of the PerFlow Tool two separate user studies were conducted. They were in their process nearly identical and the users were in both studies confronted with the same use cases and tasks. But in the second study some details at the visual scripting tool itself and at the documents handed out to the participants were tweaked based on the feedback of the first study.

At the beginning of the study, the participants received a short explanation of the PerFlow Tool. Described on about half a page, this included the overall goal of the visual scripting tool and how it can be used in smart environments. Additionally, the different visual scripting elements were explained shortly together with the functionalities offered by the tool itself comprising also about half a page. Lastly, one simple example for a rule created in the PerFlow Tool was shown. The participants then had a few minutes to familiarize themselves with the visual scripting tool and read the short introduction. As the goal of PerFlow as a pervasive middleware is to organize the information flow in many different scenarios, users may encounter the system without knowing it beforehand e.g., by holding a lecture in a new classroom. Therefore, the explanations and the introduction time for the tool was purposely short as users in real life scenarios may also encounter the system with nothing more as a short on-screen manual.

In the next step, the participants are provided with a sheet describing several scenarios and tasks they should perform by using the PerFlow Tool. In each of these tasks, they are asked to reconfigure a pervasive system according to what they would need in order to fulfill the needs of the current scenario. The scenario and tasks are discussed in more detail in Section 8.4.2. After each reconfiguration, the prototype of the PerFlow Tool saved the new set of rules allowing to check later if the users were able to create a valid configuration for the pervasive system. Lastly, the participants were asked to fill out a questionnaire and describe their experience with the visual scripting tool and how well they received it.

Based on the feedback the users gave in the first study, the PerFlow Tool itself and the description provided in the second study were refined. For the tool, the amount of visual scripting elements were reduced by condensing functionalities of redundant elements. Additionally, the naming scheme for the elements and their descriptions were improved to be less technical. The same applies to the overall user interface of the tool. These changes resulted in the design presented in Section 6.6. Further, the description of the visual scripting tool handed out to the participants was also written less technical and therefore easier to understand. Both descriptions, for the first and second study, can be seen in the Appendix C.1 and C.2.

8.4.2. Scenario and Questionnaire

After the participants had some time to familiarize themselves with the PerFlow Tool, they were handed out a description for two different scenarios. Each contained several tasks for the user to solve by using the visual scripting tool. The handouts with the description of PerFlow, the scenarios, and tasks are shown in Appendix C.1 for the first study and Appendix C.2 for the second study. During the first scenario, the participants pose as the teacher in a smart classroom and need to reconfigure the system in three different ways e.g, for a head on presentation first and then for the students to use it during group work. The second scenario placed the users in the role of an employee at an airport. Here, they needed to reconfigure a smart airport lounge in four different ways to allow, for instance, visitors to use the screen or send text messages to connected devices. For the task descriptions, the participants were provided with information on which users and devices should be able to communicate with each other and which information they want to share. Thus, they needed to figure out how these descriptions translate to possible rules and how they can be built by using the provided visual scripting elements on their own.

Regarding the data collection and the analysis of the results for a user study, we encounter many different models. Before designing the questionnaire and the actual conduction of the study, a decision for one of these models had to be made. The two most prominent and well known models are the Unified Theory of Acceptance and Use of Technology (UTAUT) model [191] and Technology Acceptance Model (TAM) [41] with its extensions TAM2 [190] and TAM3 [189]. The goal of the UTAUT model is to determine, if the user is intending to use the system by looking at the key constructs of performance expectancy, effort expectancy, social influence, and facilitating conditions. These constructs are combined with further personal informations of the user. The TAM model on the other hand focuses more on how the user perceives the usefulness and ease-of-use of the evaluated system. In most scenarios the administrator chooses a suitable pervasive middleware for the intended use and sets up the smart environment. Therefore, it is not the decision of the later end-user if the PerFlow system and its visual scripting tool is deployed in a specific use case. In the scenario of the smart lecture room the university or school decides on the system to deploy, while the lecturer reconfiguring it during the lesson is only able to use what he was provided with. Thus, for the user study the valuation of the usefulness and ease-of-use for the system by the user is more valuable as the intention of use, as the user regardless of his intention would have to use it if it is deployed in the smart environment.

Therefore, the TAM model is used as the basis for the questionnaire handed out to the participants after they completed all given tasks. The questionnaire first asked for some personal information including the gender, age, and IT knowledge. For the data acquisitionm the questionnaire contains five groups with a total of

23 Likert-scaled questions between 7 (strongly agree) and 1 (strongly disagree) and five open questions for comments. The five question categories are for the *Personal Information*, *Scenario*, *Ease of use*, *Usefulness*, and *User Interface*. The complete questionnaire for the user study can be seen in Appendix C.3.

8.4.3. Participants

In total, 43 users participated in the two studies, 20 in the first and 23 in the second. The focus was on getting a possible diverse crowd of participants. Therefore, the users were recruited from students participating in varying courses at the university and in the private environment of students and colleagues. Within the questionnaire, the participants were asked for some personal information, of which the most important one is summarized in the Table 8.1 for both studies.

	Study 1	**Study 2**
Total participants	20	23
Mean age	31.6 years	26.6 years
Male / Female	60.0% / 40.0%	43.5% / 56.5%
IT background	55.0%	45.5%

Table 8.1.: Personal information of the participants taking part in the first and second user study.

The information shows that the distribution of male to female participants is nearly 50%. While the average age of the participants was 29.1 years, the oldest user was 57 years and the youngest 19 years old. Of all participants 5 have an educational degree below highschool, 10 a highschool degree, and 28 have an academic degree. Most importantly for the user study about half of the participants do not work or study in an IT related field. As the PerFlow Tool is aiming to non professional users, like the lecturer in the scenario introduced in Section 4.1, the feedback and assessment of users without knowledge in pervasive systems or programming is highly valued.

8.4.4. Results

After describing the structure of the user study, the participants, and the tasks they had to fulfill, the next section will focus on the results obtained through the questionnaire. As already discussed, two studies were conducted and the PerFlow Tool, as well as the documentation, was improved in between based on the feedback of the first study. While it was certainly helpful to evaluate the Likert-scale questions, the most important feedback for the improvements came in the form of the open questions where many participants gave useful suggestions. Additionally, the participants were monitored while solving the tasks. During the overhaul of the PerFlow Tool, the functionalities itself and the process of creating new rules remained unchanged. The changes can be categorized in *improved naming scheme*, *simplified visual scripting language*, and *improved error handling*. Many participants, especially these without IT knowledge, showed problems understanding some of the terms used for the visual scripting language and tool. Therefore, a less technical *naming scheme* was introduced, which means that e.g., the "Input Device" and "Output Device" was renamed to "Sender" and "Receiver". Further, in the first iteration of the PerFlow Tool the *visual scripting language* was more complex and contained e.g., redundant elements which could be used to create the same outcome in different ways. Some users found this confusing, which led to a reduction of elements. Lastly, the *error handling* was improved by identifying possible error sources and eliminating them. This has been done by e.g., changing text fields to drop down boxes or not allowing certain combinations of elements. Due to the simplification of the user interface and the overall more understandable naming scheme it was also possible to slightly shorten the documentation for the participants without introducing obstacles for the users.

During the study the PerFlow Tool saved the created rules for each user and tasks. This allowed the evaluation of how the users performed and if it would have been possible to use the created configurations in a real system. By looking at the results, the effect of the improvements between the studies is visible. During the first study seven out of the 20 participants created in at least one task rules with minor to severe errors, which would have led to a faulty system. Even so the handout with the introduction for the visual scripting tool was simplified and reduced in length for the second study the participants produced way better results. Only two out of the 23 participants created the same kind of errors, while one other user made a small error, which would have easily been caught with an improved error handling while saving the configuration.

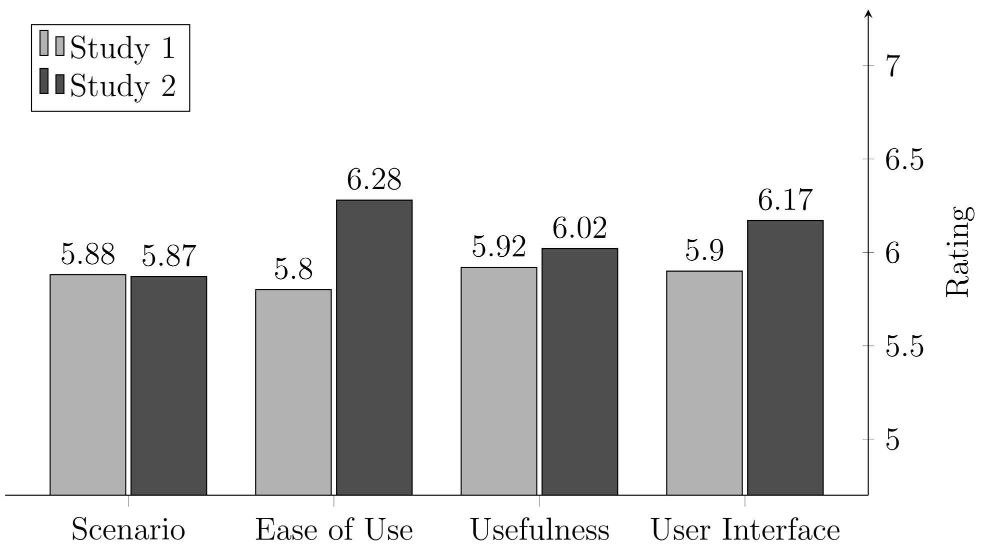


Figure 8.8.: Comparison of the average rating of the first and second user study in the different categories. The values show an overall increase in the second study.

The average results for the four categories of Likert-scaled questions can be seen in Figure 8.8. Additionally, the results for both studies can be compared. As the description of the PerFlow Tool was simplified and less information was provided, the results showed that the participants rated the scenario, helping material, and documentation lower in the second study. Especially the question on how satisfied the participants were with the provided helping material dropped from an average 6.10 to a 5.65 rating. Nonetheless, the ratings in all other categories increased and were between 6 (moderately agree) and 7 (strongly agree) for the second study. The strong improvement is visible in the average values, where especially an increase from 5.80 to 6.28 for the the *Ease of use* can be seen and the *User Interface* is also evaluated as more intuitive. Especially noteworthy is that the result for the statement "*I find the application easy to use.*" was raised by 0.6 points to 6.35 and for "*The function of the elements is clear.*" the result increased by 0.81 points to 6.26 in average. In both studies the participants saw the overall appeal of the application and valuated the *Usefulness* of the PerFlow Tool for the scenarios with 5.97 in average. Even though the participants were already quite satisfied with the visual scripting tool in the first study, some small tweaks without altering the functionalities made a large impact in the perceived

Ease of use. Appendix C.4 shows the average results for each question for the first and second user study.

8.5. Summary

In this chapter, the evaluation of the prototype of the PerFlow system, including the PerFlow Middleware, PerFlow Tool, and PerFlow Virtual Extension, was presented. The evaluation was split into four parts. First, a qualitative evaluation was performed introducing the PerLE testbed for pervasive classrooms. Second, the implementation effort for developers while using the PerFlow Middleware was examined. Third, four different performance measurements were conducted looking at the time needed to reconfigure the PerFlow system and the overhead introduced by the middleware, the virtual extension, and the consensus algorithm. Fourth, two user studies were discussed showing the usability and ease of use of the PerFlow Tool.

Requirement	**Discussion**
R_{F1} - Information Exchange	PerLE
R_{F2} - Runtime Reconfiguration	PerLE
R_{F3} - Bundling Information Flow	PerLE
R_{F4} - Heterogeneity Support	PerLE
R_{F5} - Device Management	PerLE
R_{F6} - Access Control	PerLE
R_{F7} - Remote Access	PerLE
R_{NF1} - Responsiveness	Performance Measurements
R_{NF2} - Fault Tolerance	User Study
R_{NF3} - Generalizability	PerLE & User Study
R_{NF4} - Usability	User Study & Implementation Effort
R_{NF5} - Extensibility	PerLE

Table 8.2.: The requirements for PerFlow introduced in Chapter 4 and where they are discussed in the evaluation.

In Table 8.2 the requirements introduced in Chapter 4 are shown together with the associated part of the evaluation. The functional requirements R_{F1} - R_{F7} were discussed in Section 8.1 and the fulfillment was shown at the example of the PerLE testbed. Further, the conducted performance measurements showed that it is possible to develop applications in the PerFlow system which are responsive (R_{NF1}). The user studies resulted in a high usability (R_{NF4}) and after some small adjustments most of the participants in the second study made no mistakes while creating the rules (R_{NF2}). Additionally, in Section 8.2 we could show that a high usability (R_{NF4}) of the PerFlow Middleware is also given for developers. During the evaluation three possible use cases for smart classrooms, meeting rooms, and airport lounges were discussed, which support the argument for a generalizable system (R_{NF3}). Lastly, during the development of the PerLE testbed several new file formats were introduced for the communication via the PerFlow Middleware e.g., PDF or voice recordings, showing the extensibility of the system (R_{NF5}).

9. Conclusion and Outlook

In the previous chapter, we discussed the evaluation of the prototype for the PerFlow system, including the PerFlow Tool and PerFlow Virtual Extension. The evaluation was comprised of a proof of concept with the PerLE testbed, an analysis of the implementation effort, several performance measurements, and a two stage user study for the visual scripting language and tool. In this chapter we close the thesis with a conclusion and give an outlook on possible future work.

9.1. Conclusion

With the increasing number of smart devices in today's environments many people are encountering them in a daily basis. This is not only true for people directly working in the field of information technology or people with a high interest in novel technologies, but also for many others. Often, users are even forced to interact with smart devices, if, for instance, their employer deploys a new smart meeting room or their school decides to introduce intelligent classrooms. This ubiquity of smart devices bears several challenges. While smart environments are often set up by administrators and offer possibilities to adapt themselves to the surrounding context, it is impossible to predict every scenario a user could encounter. Therefore, end users need the possibility to influence the behavior of the system on their own. This is especially challenging for users that are not familiar with smart devices. Further, while people start relying on their smart devices and use them for help with their daily task, these systems are often getting critical to the work performance of some users. This leads to the fact, that users may need access to their systems even while they are not physically on location.

To tackle these challenges and answer the research questions postulated in Section 1.2, this thesis presented the PerFlow system. The design of the system is split into three major parts, each supporting one of the research questions. First, the PerFlow Middleware is responsible for handling the communication in a pervasive system and allows the reconfiguration of the information flow during runtime. To achieve this, application developers are not required to define themselves to which other applications or devices they want theirs to connect. Instead, they need to inform the middleware which information their application is able to provide or receive. The middleware is then responsible to distribute the data. This is done by the middleware according to configurations provided by users. After handing over a new configuration, the middleware is also responsible for validating and distributing it throughout the system. Second, with the PerFlow Tool we offer end users an easy to use visual scripting tool to enable them to create their own configurations for the middleware and thus reconfigure the system to their needs. Therefore, an own visual scripting language was introduced. The language is designed to create rules for the information flow in a pervasive system. Users submit new sets of rules and the middleware interprets them and reconfigures the system accordingly. Third, the PerFlow Virtual Extension gives developers the possibility to transfer their services and applications to a virtual environment. Which gives end users the ability to have a comparable user experience remotely as the users physically on location. The virtual extension utilizes an existing game engine and provides a proxy to enable the communication of virtual applications with the middleware.

Finally, the thesis discusses the implementation of a prototype for the complete PerFlow system and provides an in depth evaluation. The evaluation was conducted in four steps. First, showing with a proof of concept that it is possible to reconfigure the pervasive system during the runtime. Therefore, the PerLE testbed was introduced, showing how PerFlow could be used in a learning environment scenario. Second, the evaluation showed that the use of the PerFlow Middleware reduces the implementation effort for application developers. Third, performance measurements led to the conclusion that the PerFlow Middleware does not introduce a noticeable overhead compared to the used communication middleware. Additionally, remote users are also experiencing no overhead when using the PerFlow Virtual Extension. Fourth, a two stage user study demonstrated that it is possible for unexperienced users to configure a pervasive system with the introduced PerFlow Tool.

9.2. Outlook

During the design and development of the PerFlow system we encountered several research challenges. These provide a promising starting point for possible future work.

While the PerFlow Middleware showed good performance with BASE [16] as the chosen communication middleware, it may be worth analyzing different middlewares. In this case, it would be of special interest to look at middlewares with different communication models. As BASE uses remote procedure calls, a comparison with middlewares based on publish-subscribe or tuple spaces could lead to new insights.

Further, as the PerFlow Tool can be easily exchanged for other visual scripting tools, it provides the possibility to compare different visual scripting languages. The chosen data flow based visual alphabet showed in the user study a good usability for the configuration of the information flow. Nonetheless, a comparison with other alphabets, e.g., an iconic, graph, or box based alphabet, could lead to further insights. This would require to first analyze how the different alphabets could be applied to the use case of a pervasive system and then design and develop a new language and visual scripting tool.

Additionally, a direct integration of the PerFlow Middleware into the architecture of a game engine could lead to a significant performance improve. With such an integration the separate proxy for the PerFlow Virtual Extension and thus one extra serialization step could be saved. But this would require a cooperation with one of the commercial developers to get access to the low level architecture of their game engine.

Lastly, a large and long term field study for the complete PerFlow system would be desirable. By deploying the system for instance at a university or school, it would be possible to analyze how lectures benefit from pervasive systems and how lecturers and students rate the usability.

Bibliography

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In H. Gellersen, editor, *Handheld and Ubiquitous Computing, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307. Springer, 1999.

[2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In D. Kotz and J. Wilkes, editors, *Proceedings of the 17th ACM Symposium on Operating System Principles, SOSP 1999, Kiawah Island Resort, near Charleston, South Carolina, USA, December 12-15, 1999*, pages 186–201. ACM, 1999.

[3] Adobe Inc. Adobe Connect web conferencing software. https://www.adobe.com/products/adobeconnect.html. Online; accessed: October 02, 2019.

[4] S. Ahmed, M. Sharmin, and S. I. Ahamed. A smart meeting room with pervasive computing technologies. In L. Chung and Y. Song, editors, *Proceedings of the 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2005), May 23-25, 2005, Towson, Maryland, USA*, pages 366–371. IEEE Computer Society, 2005.

[5] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser. Mundocore: A light-weight infrastructure for pervasive computing. *Pervasive and Mobile Computing*, 3(4):332–361, 2007.

[6] A. J. Albrecht and J. E. G. Jr. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Trans. Software Eng.*, 9(6):639–648, 1983.

[7] Amazon Web Services, Inc. AWS IoT. https://aws.amazon.com/de/iot/. Online; accessed: September 26, 2019.

[8] Apple Inc. Apple HomeKit. https://www.apple.com/de/ios/home/. Online; accessed: September 26, 2019.

[9] Apple Inc. Apple Human Interface Guidelines. `https://developer.apple.com/design/human-interface-guidelines/`. Online; accessed: August 17, 2019.

[10] Artifex Software, Inc. Ghostscript. `https://www.ghostscript.com/`. Online; accessed: August 6, 2019.

[11] K. Ashton. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[12] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[13] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *IJAHUC*, 2(4):263–277, 2007.

[14] R. Bardohl. A visual environment for visual languages. *Sci. Comput. Program.*, 44(2):181–203, 2002.

[15] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM - A component system for pervasive computing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*, pages 67–76. IEEE Computer Society, 2004.

[16] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. BASE - A micro-broker-based middleware for pervasive computing. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), March 23-26, 2003, Fort Worth, Texas, USA*, pages 443–451. IEEE Computer Society, 2003.

[17] S. Benford and L. E. Fahlén. A spatial model of interaction in large virtual environments. In *Third European Conference on Computer Supported Cooperative Work, ECSCW'93, Milano, Italy, September 13-17, 1993, Proceedings*, page 107. Kluwer Academic Publishers, 1993.

[18] S. Benford, C. Greenhalgh, G. Reynard, C. Brown, and B. Koleva. Understanding and constructing shared spaces with mixed-reality boundaries. *ACM Trans. Comput.-Hum. Interact.*, 5(3):185–223, 1998.

[19] C. Böhm and G. Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, 1966.

[20] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan Education UK, 1976.

[21] J. Botev, A. Höhfeld, H. Schloss, I. Scholtes, P. Sturm, and M. Esch. The hyperverse: concepts for a federated and torrent-based '3d web'. *IJAMC*, 2(4):331–350, 2008.

[22] A. Bröring, S. Schmid, C. K. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. L. Phuoc, J. Mitic, D. Anicic, and E. Teniente. Enabling iot ecosystems through platform interoperability. *IEEE Software*, 34(1):54–61, 2017.

[23] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Commun.*, 4(5):58–64, 1997.

[24] M. M. Burnett and A. L. Ambler. A declarative approach to event-handling in visual programming languages. In *Proceedings of the 1992 IEEE Workshop on Visual Languages, September 15-18, 1992, Seattle, Washington, USA*, pages 34–40. IEEE Computer Society, 1992.

[25] H. E. Byun and K. Cheverst. Supporting proactive 'intelligent' behaviour: the problem of uncertainty. In *Proceedings of the UM03 Workshop on User Modeling for Ubiquitous Computing*, pages 17–25, 2003.

[26] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: context-aware reflective middleware system for mobile applications. *IEEE Trans. Software Eng.*, 29(10):929–945, 2003.

[27] D. Carlson and A. Schrader. Dynamix: An open plug-and-play context framework for android. In *3rd IEEE International Conference on the Internet of Things, IOT 2012, Wuxi, Jiangsu Province, China, October 24-26, 2012*, pages 151–158. IEEE, 2012.

[28] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of Computer Science, Univ. of Utah, 1974.

[29] A. T. S. Chan and S. N. Chuang. MobiPADS: A reflective middleware for context-aware mobile computing. *IEEE Trans. Software Eng.*, 29(12):1072–1085, 2003.

[30] C. Chang, S. Ling, and S. Krishnaswamy. ProMWS: Proactive mobile web service provision using context-awareness. In *Ninth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2011, 21-25 March 2011, Seattle, WA, USA, Workshop Proceedings*, pages 69–74. IEEE Computer Society, 2011.

[31] S. Chang. Visual languages: A tutorial and survey. *IEEE Software*, 4(1):29–39, 1987.

[32] S. Chang, M. J. Tauber, B. Yu, and J. Yu. A visual language compiler. *IEEE Trans. Software Eng.*, 15(5):506–525, 1989.

[33] K. Charntaweekhun and S. Wangsiripitak. Visual programming using flowchart. In *2006 International Symposium on Communications and Information Technologies*, pages 1062–1065. IEEE, 2006.

[34] H. Chen. *An intelligent broker architecture for pervasive context-aware systems.* PhD thesis, University of Maryland, Baltimore County, 2004.

[35] S. Chetan, J. Al-Muhtadi, R. Campbell, and M. D. Mickunas. Mobile gaia: a middleware for ad-hoc pervasive computing. In *Second IEEE Consumer Communications and Networking Conference, 2005. CCNC. 2005*, pages 223–228, Jan 2005.

[36] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In T. Turner and G. Szwillus, editors, *Proceedings of the CHI 2000 Conference on Human factors in computing systems, The Hague, The Netherlands, April 1-6, 2000.*, pages 17–24. ACM, 2000.

[37] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, and S. Zachariadis. The RUNES middleware: a reconfigurable component-based approach to networked embedded systems. In *Proceedings of the IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications, Berlin, Germany, September 11-14, 2005*, pages 806–810. IEEE, 2005.

[38] G. Costagliola, A. D. Lucia, S. Orefice, and G. Polese. A classification framework to support the design of visual languages. *J. Vis. Lang. Comput.*, 13(6):573–600, 2002.

[39] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems - concepts and designs (3. ed.)*. International computer science series. Addison-Wesley-Longman, 2002.

[40] N. Davies, K. Mitchell, K. Cheverst, and G. Blair. Developing a context sensitive tourist guide. In *1st Workshop on Human Computer Interaction with Mobile Devices, GIST Technical Report G98-1*, volume 1, 1998.

[41] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw. User acceptance of computer technology: A comparison of two theoretical models. *Management Science*, 35(8):982–1003, 1989.

[42] M. Davis. Media streams: An iconic visual language for video annotation. In *Proceedings of the 1993 IEEE Workshop on Visual Languages, August 24-27, 1993, Bergen, Norway*, pages 196–202. IEEE Computer Society, 1993.

[43] A. K. Dey. Context-aware computing: The cyberdesk project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, 1998.

[44] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[45] Dom4j. Flexible XML framework for Java. `https://dom4j.github.io/`. Online; accessed: August 6, 2019.

[46] Easy Virtual Fair. Easy Virtual Fair. Online; accessed: October 02, 2019.

[47] W. K. Edwards, M. W. Newman, J. Z. Sedivy, and S. Izadi. Challenge: recombinant computing and the speakeasy approach. In I. F. Akyildiz, J. Y. Lin, R. Jain, V. Bharghavan, and A. T. Campbell, editors, *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking, MOBICOM 2002, Atlanta, Georgia, USA, September 23-28, 2002*, pages 279–286. ACM, 2002.

[48] S. R. Ellis. Nature and origins of virtual environments: a bibliographical essay. In R. M. BAECKER, J. GRUDIN, W. A. BUXTON, and S. GREEN-

BERG, editors, *Readings in Humanâ€“Computer Interaction*, Interactive Technologies, pages 913 – 932. Morgan Kaufmann, 1995.

[49] C. Escoffier, S. Chollet, and P. Lalanda. Lessons learned in building pervasive platforms. In *11th IEEE Consumer Communications and Networking Conference, CCNC 2014, Las Vegas, NV, USA, January 10-13, 2014*, pages 7–12. IEEE, 2014.

[50] C. Escoffier, R. S. Hall, and P. Lalanda. ipojo: an extensible service-oriented component framework. In *2007 IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA*, pages 474–481. IEEE Computer Society, 2007.

[51] Facebook Technologies, LLC. Oculus Rift S. `https://www.oculus.com/rift-s/`. Online; accessed: August 6, 2019.

[52] Facebook Technologies, LLC. Oculus Venues. https://www.oculus.com/experiences/go/1555304044520126/. Online; accessed: October 02, 2019.

[53] J. Feng and J. Li. Google protocol buffers research and application in online game. In *IEEE Conference Anthology*, pages 1–4, Jan 2013.

[54] W. A. Fetter. A progression of human figures simulated by computer graphics. *IEEE Computer Graphics and Applications*, (9):9–13, 1982.

[55] H. W. Franke. *Computer graphics - computer art.* Springer Science & Business Media, 2012.

[56] E. Frécon and M. Stenius. DIVE: a scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91–100, 1998.

[57] D. Garlan, D. P. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.

[58] E. P. Glinert. Out of flatland: towards 3-d visual programming. In S. A. Szygenda, editor, *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 292–299. ACM, 1987.

[59] E. J. Golin and S. P. Reiss. The specification of visual language syntax. *J. Vis. Lang. Comput.*, 1(2):141–157, 1990.

[60] Google Inc. Google Android Design Guidelines. `https://developer.android.com/docs/quality-guidelines`. Online; accessed: August 17, 2019.

[61] Google Inc. Gson library. `https://github.com/google/gson`. Online; accessed: August 6, 2019.

[62] Google Inc. Protocol Buffers. `https://developers.google.com/protocol-buffers/`. Online; accessed: August 6, 2019.

[63] M. L. Graf. A visual environment for the design of distributed systems. In *Visual Languages and Applications*, pages 53–67. Springer, 1990.

[64] C. Greenhalgh and S. Benford. Virtual reality tele-conferencing: Implementation and experience. In *Fourth European Conference on Computer Supported Cooperative Work, ECSCW'95, Stockholm, Sweden, September 11-15, 1995, Proceedings*, page 163. Kluwer Academic Publishers, 1995.

[65] J. Gregory. *Game engine architecture.* AK Peters/CRC Press, 2017.

[66] R. Grimm. One.world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, 3(3):22–30, 2004.

[67] R. Grimm, J. Davis, E. Lemar, A. MacBeth, S. Swanson, T. E. Anderson, B. N. Bershad, G. Borriello, S. D. Gribble, and D. Wetherall. System support for pervasive applications. *ACM Trans. Comput. Syst.*, 22(4):421–486, 2004.

[68] T. Gu, H. K. Pung, and D. Zhang. A service-oriented middleware for building context-aware services. *J. Network and Computer Applications*, 28(1):1–18, 2005.

[69] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Comp. Syst.*, 29(7):1645–1660, 2013.

[70] S. Gundlach and M. K. Martin. *Mastering CryENGINE.* Packt Publishing Ltd, Birmingham, UK, 2014.

[71] M. Handte, C. Becker, and K. Rothermel. Peer-based automatic configuration of pervasive applications. *Int. J. Pervasive Computing and Communications*, 1(4):251–264, 2005.

[72] M. Handte, G. Schiele, V. Majuntke, C. Becker, and P. J. Marrón. 3pc: System support for adaptive peer-to-peer pervasive computing. *TAAS*, 7(1):10:1–10:19, 2012.

[73] P. Hanrahan and W. Krueger. Reflection from layered surfaces due to subsurface scattering. In M. C. Whitton, editor, *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1993, Anaheim, CA, USA, August 2-6, 1993*, pages 165–174. ACM, 1993.

[74] T. R. Hansen and J. E. Bardram. Activetheatre - A collaborative, event-based capture and access system for the operating theatre. In M. Beigl, S. S. Intille, J. Rekimoto, and H. Tokuda, editors, *UbiComp 2005: Ubiquitous Computing, 7th International Conference, UbiComp 2005, Tokyo, Japan, September 11-14, 2005, Proceedings*, volume 3660 of *Lecture Notes in Computer Science*, pages 375–392. Springer, 2005.

[75] HashiCorp. HashiCorp Consul. `https://www.hashicorp.com/products/consul/`. Online; accessed: July 18, 2019.

[76] F. Heger, G. Schiele, R. Süselbeck, L. Itzel, and C. Becker. Scalability in peer-to-peer-based mmves: The continuous events approach. In *2012 IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, January 14-17, 2012*, pages 629–633. IEEE, 2012.

[77] S. Helal, W. C. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: A programmable pervasive space. *IEEE Computer*, 38(3):50–60, 2005.

[78] K. Herrmann, K. Rothermel, G. Kortuem, and N. Dulay. Adaptable pervasive flows - an emerging technology for pervasive adaptation. In *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2008, Workshops Proceedings, October 20-24, 2008, Venice, Italy*, pages 108–113. IEEE Computer Society, 2008.

[79] C. K. Hess, M. Román, and R. H. Campbell. Building applications for ubiquitous computing environments. In F. Mattern and M. Naghshineh, editors, *Pervasive Computing, First International Conference, Pervasive 2002, Zürich, Switzerland, August 26-28, 2002, Proceedings*, volume 2414 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2002.

[80] D. D. Hils. Visual languages and computing survey: Data flow visual programming languages. *J. Vis. Lang. Comput.*, 3(1):69–101, 1992.

[81] M. Hirakawa, M. Tanaka, and T. Ichikawa. An iconic programming system, HI-VISUAL. *IEEE Trans. Software Eng.*, 16(10):1178–1184, 1990.

[82] S. Holloway and C. Julien. The case for end-user programming of ubiquitous computing environments. In G. Roman and K. J. Sullivan, editors, *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, pages 167–172. ACM, 2010.

[83] J. Hong, E. Suh, J. Kim, and S. Kim. Context-aware system for proactive personalized service based on context history. *Expert Syst. Appl.*, 36(4):7448–7457, 2009.

[84] J. Hong, E. Suh, and S. Kim. Context-aware systems: A literature review and classification. *Expert Syst. Appl.*, 36(4):8509–8522, 2009.

[85] HTC Corporation. HTC Vive. `https://www.vive.com/de/product/`. Online; accessed: August 6, 2019.

[86] S. Hu, J. Chen, and T. Chen. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, 2006.

[87] S.-Y. Hu. Spatial publish subscribe. In *Proc. of IEEE Virtual Reality (IEEE VR) workshop, Massively Multiuser Virtual Environment (MMVEâ€™09)*, 2009.

[88] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[89] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In P. Barham and T. Roscoe, editors, *2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010*. USENIX Association, 2010.

[90] IFTTT, Inc. IFTTT: If this then that. https://ifttt.com/. Online; accessed: September 26, 2019.

[91] ILIAS open source e-Learning Society. ILIAS open source learning management system. https://www.ilias.de/en/. Online; accessed: October 02, 2019.

[92] A. Indraprastha and M. Shinozaki. The investigation on using unity3d game engine in urban design study. *Journal of ICT Research and Applications*, 3(1):1–18, 2009.

[93] H. Ingo. Four modifications for the Raft consensus algorithm. Technical report, 2015.

[94] S. Izadi, A. Agarwal, A. Criminisi, J. M. Winn, A. Blake, and A. W. Fitzgibbon. C-slate: A multi-touch and object recognition system for remote collaboration using horizontal surfaces. In *Second IEEE International Workshop on Horizontal Interactive Human-Computer Systems (Tabletop 2007), October 10-12 2007, Newport, Rhode Island, USA*, pages 3–10. IEEE Computer Society, 2007.

[95] M. Jeronimo and J. Weast. *UPnP design by example*, volume 158. Intel Press, 2003.

[96] JGraphT. JGraphT library of graph theory data structures and algorithms. `https://jgrapht.org/javadoc/`. Online; accessed: August 20, 2019.

[97] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002.

[98] JS Foundation. Node-RED. https://nodered.org/. Online; accessed: August 17, 2019.

[99] JSON. JSON Introduction. `http://json.org/`. Online; accessed: August 6, 2019.

[100] F. P. Junqueira, B. C. Reed, and M. Serafini. Zab: High-performance broadcast for primary-backup systems. In *Proceedings of the 2011 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2011, Hong Kong, China, June 27-30 2011*, pages 245–256. IEEE Compute Society, 2011.

[101] P. Kauff and O. Schreer. An immersive 3d video-conferencing system using shared virtual team user environments. In W. Broll, C. Greenhalgh, and

E. F. Churchill, editors, *Proceedings of the 4th International Conference on Collaborative Virtual Environments 2002, Bonn, Germany, September 30 - October 02, 2002*, pages 105–112. ACM, 2002.

[102] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. S. Magalhães, and R. H. Campbell. Monitoring, security, and dynamic configuration with the *dynamicTAO* reflective ORB. In J. S. Sventek and G. Coulson, editors, *Middleware 2000, IFIP/ACM International Conference on Distributed Systems Platforms, New York, NY, USA, April 4-7, 2000, Proceedings*, volume 1795 of *Lecture Notes in Computer Science*, pages 121–143. Springer, 2000.

[103] C. D. Kounavis, A. E. Kasimati, and E. D. Zamani. Enhancing the tourism experience through mobile augmented reality: Challenges and prospects. *International Journal of Engineering Business Management*, 4:10, 2012.

[104] P. Lalanda, C. Hamon, C. Escoffier, and T. Leveque. icasa, a development and simulation environment for pervasive home applications. In *Proc. CCNC*, 2014.

[105] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[106] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[107] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[108] P. A. Laplante. *Requirements engineering for software and systems.* Auerbach Publications, 2017.

[109] R. Lea, Y. Honda, and K. Matsuda. Virtual society: Collaboration in 3d spaces on the internet. *Computer Supported Cooperative Work*, 6(2/3):227–250, 1997.

[110] G. Lepouras and C. Vassilakis. Virtual museums for all: employing game technology for edutainment. *Virtual Reality*, 8(2):96–106, 2004.

[111] M. Lewis and J. Jacobson. Game engines in scientific research - introduction. *Commun. ACM*, 45(1):27–31, 2002.

[112] Linden Research, Inc. Second Life. `https://secondlife.com`. Online; accessed: October 02, 2019.

[113] B. Liskov and J. Cowling. Viewstamped replication revisited. Technical report, MIT, 2012.

[114] A. D. Lucia, R. Francese, I. Passero, and G. Tortora. Development and evaluation of a virtual campus on second life: The case of seconddmi. *Computers & Education*, 52(1):220–233, 2009.

[115] V. Majuntke, S. VanSyckel, D. Schäfer, C. Krupitzer, G. Schiele, and C. Becker. COMITY: coordinated application adaptation in multi-platform pervasive systems. In *2013 IEEE International Conference on Pervasive Computing and Communications, PerCom 2013, San Diego, CA, USA, March 18-22, 2013*, pages 11–19. IEEE Computer Society, 2013.

[116] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *TOCE*, 10(4):16:1–16:15, 2010.

[117] C. Marinagi, P. Belsis, and C. Skourlas. New directions for pervasive computing in logistics. *Procedia - Social and Behavioral Sciences*, 73:495 – 502, 2013.

[118] J. Martin. *Managing the Data Base Environment.* A James Martin book. Pearson Education, Limited, 1983.

[119] meetyoo conferencing GmbH. meetyoo conferencing. https://meetyoo.com/en/. Online; accessed: October 02, 2019.

[120] Microsoft Corporation. Azure IoT Hub. https://azure.microsoft.com/de-de/services/iot-hub/. Online; accessed: September 26, 2019.

[121] Microsoft Corporation. Microsoft Teams. https://products.office.com/en-us/microsoft-teams/group-chat-software. Online; accessed: October 02, 2019.

[122] Microsoft Corporation. Microsoft Universal Windows Platform. `https://docs.microsoft.com/en-us/windows/uwp/`. Online; accessed: August 6, 2019.

[123] Microsoft Corporation. Microsoft UWP Design Guidelines. `https://docs.microsoft.com/en-us/windows/uwp/design/`. Online; accessed: August 17, 2019.

[124] G. S. P. Miller. Efficient algorithms for local and global accessibility shading. In D. Schweitzer, A. S. Glassner, and M. Keeler, editors, *Proceedings of the 21th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994, Orlando, FL, USA, July 24-29, 1994*, pages 319–326. ACM, 1994.

[125] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[126] K. L. Morse. Interest management in large-scale distributed simulations. Technical report, Information and Computer Science, University of California, Irvine, 1996.

[127] M. A. Musen, L. M. Fagan, and E. H. Shortliffe. Graphical specification of procedural knowledge for an expert system. *Expert systems: the user interface*, pages 15–35, 1988.

[128] B. A. Myers. Visual programming, programming by example, and program visualization: a taxonomy. In *ACM sigchi bulletin*, volume 17, pages 59–66. ACM, 1986.

[129] B. A. Myers. Taxonomies of visual programming and program visualization. *J. Vis. Lang. Comput.*, 1(1):97–123, 1990.

[130] B. A. Myers, J. F. Pane, and A. J. Ko. Natural programming languages and environments. *Commun. ACM*, 47(9):47–52, 2004.

[131] J. Naber, C. Krupitzer, and C. Becker. Transferring an interactive display service to the virtual reality. In *2017 IEEE International Conference on Smart Computing, SMARTCOMP 2017, Hong Kong, China, May 29-31, 2017*, pages 1–8. IEEE Computer Society, 2017.

[132] J. Naber, M. Pfannemüller, J. Edinger, and C. Becker. Perflow: Configuring the information flow in a pervasive middleware via visual scripting. In *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous*. ACM, 2019. to be puplished.

[133] J. Naber, D. Schäfer, S. VanSyckel, and C. Becker. Interactive display services for smart environments. In Y. Wu, G. Min, N. Georgalas, J. Hu,

L. Atzori, X. Jin, S. A. Jarvis, L. C. Liu, and R. A. Calvo, editors, *15th IEEE International Conference on Computer and Information Technology, CIT 2015; 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015; 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015; 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015, Liverpool, United Kingdom, October 26-28, 2015*, pages 2157–2164. IEEE, 2015.

[134] J. Naber, S. Schmitz, and C. Becker. Perle: A testbed for pervasive middlewares in learning environments. In *IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019, Kyoto, Japan, March 11-15, 2019*, pages 474–479. IEEE, 2019.

[135] B. J. Nelson. *Remote Procedure Call.* PhD thesis, Pittsburgh, PA, USA, 1981.

[136] M. W. Newman, J. Z. Sedivy, C. Neuwirth, W. K. Edwards, J. I. Hong, S. Izadi, K. Marcelo, and T. F. Smith. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In B. Verplank, A. G. Sutcliffe, W. E. Mackay, J. Amowitz, and W. W. Gaver, editors, *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, DIS 2002, London, England, UK, June 25-28, 2002*, pages 147–156. ACM, 2002.

[137] NextVR Inc. NextVR: Virtual Reality Events. https://nextvr.com/. Online; accessed: October 02, 2019.

[138] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. Boehm. A sloc counting standard. Technical report, Cocomo II Forum, 2007.

[139] M. Nidd. Service discovery in deapspace. *IEEE Personal Commun.*, 8(4):39–45, 2001.

[140] D. A. Norman. *The invisible computer - why good products can fail, the personal computer is so complex, and information appliances are the solution.* MIT Press, 1999.

[141] NSpeex. Speex for .NET. `https://github.com/aijingsun6/NSpeex`. Online; accessed: August 6, 2019.

[142] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162, 2009.

[143] OASIS. MQTT Specification Version 3.1.1. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`, 2018. Online; accessed: August 6, 2019.

[144] D. Ongaro. *Consensus: Bridging theory and practice*. PhD thesis, Stanford University, 2014.

[145] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In G. Gibson and N. Zeldovich, editors, *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.*, pages 305–319. USENIX Association, 2014.

[146] openHAB Foundation e.V. openHAB. https://www.openhab.org/. Online; accessed: September 26, 2019.

[147] Oracle Inc. Java SE 9. `https://docs.oracle.com/javase/9/docs/api/overview-summary.html`. Online; accessed: August 6, 2019.

[148] Oracle Inc. SAX parser for Java. `https://docs.oracle.com/javase/9/docs/api/javax/xml/parsers/SAXParser.html`. Online; accessed: August 6, 2019.

[149] J. K. Ousterhout. Scripting: Higher-level programming for the 21st century. *IEEE Computer*, 31(3):23–30, 1998.

[150] F. Paganelli and D. Giuli. An ontology-based system for context-aware and configurable services to support home-based continuous care. *IEEE Trans. Information Technology in Biomedicine*, 15(2):324–333, 2011.

[151] L. Pantel and L. C. Wolf. On the suitability of dead reckoning schemes for games. In L. C. Wolf, editor, *Proceedings of the 1st Workshop on Network and System Support for Games, NETGAMES 2002, Braunschweig, Germany, April 16-17, 2002, 2003*, pages 79–84. ACM, 2002.

[152] M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *4th International Conference on Web Information Systems Engineering, WISE 2003, Rome, Italy, December 10-12, 2003*, pages 3–12. IEEE Computer Society, 2003.

[153] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: a research roadmap. *Int. J. Cooperative Inf. Syst.*, 17(2):223–255, 2008.

[154] D. J. Patterson, O. Etzioni, D. Fox, and H. Kautz. Intelligent ubiquitous computing to support alzheimer's patients: Enabling the cognitively disabled. In *Adjunct Proceedings*, page 21, 2002.

[155] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[156] P. Petridis, I. Dunwell, S. de Freitas, and D. Panzoli. An engine selection methodology for high fidelity serious games. In K. Debattista, M. D. Dickey, A. Proença, and L. P. Santos, editors, *Second International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2010, Braga, Portugal, March 25-26, 2010*, pages 27–34. IEEE Computer Society, 2010.

[157] Protobuf-Net. Protocol Buffers library for idiomatic .NET. `https://github.com/protobuf-net/protobuf-net`. Online; accessed: August 6, 2019.

[158] P. Raverdy, V. Issarny, R. Chibout, and A. de La Chapelle. A multi-protocol approach to service discovery and access in pervasive environments. In H. Ahmadi and T. L. Porta, editors, *3rd Annual International ICST Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MOBIQUITOUS 2006, San Jose, California, USA, July 17-21, 2006*, pages 1–9. IEEE Computer Society, 2006.

[159] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing*, 9(2):177–200, 2013.

[160] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, and Y. B. Kafai. Scratch: programming for all. *Commun. ACM*, 52(11):60–67, 2009.

[161] L. Ricci and E. Carlini. Distributed virtual environments: From client server to cloud and P2P architectures. In W. W. Smari and V. Zeljkovic, editors, *2012 International Conference on High Performance Computing & Simulation, HPCS 2012, Madrid, Spain, July 2-6, 2012*, pages 8–17. IEEE, 2012.

[162] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[163] D. Romero, G. Hermosillo, A. Taherkordi, R. Nzekwa, R. Rouvoy, and F. Eliassen. The digihome service-oriented platform. *Softw., Pract. Exper.*, 43(10):1205–1218, 2013.

[164] F. M. Roth, M. Pfannemüller, C. Becker, and P. Lalanda. An interoperable notification service for pervasive computing. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2018, Athens, Greece, March 19-23, 2018*, pages 842–847. IEEE Computer Society, 2018.

[165] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. O. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz. MUSIC: middleware support for self-adaptation in ubiquitous and service-oriented environments. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, volume 5525 of *Lecture Notes in Computer Science*, pages 164–182. Springer, 2009.

[166] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In *Computer applications in archaeology*. Tempus Reparatum, 1998.

[167] D. Saha and A. Mukherjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer*, 36(3):25–31, 2003.

[168] M. Satyanarayanan. Mobile computing: where's the tofu? *Mobile Computing and Communications Review*, 1(1):17–21, 1997.

[169] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Commun.*, 8(4):10–17, 2001.

[170] M. Satyanarayanan. From the editor in chief: The many faces of adaptation. *IEEE Pervasive Computing*, 3(3):4–5, 2004.

[171] B. N. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *First Workshop on Mobile Computing Systems and Applications, WMCSA 1994, Santa Cruz, CA, USA, December 8-9, 1994*, pages 85–90. IEEE Computer Society, 1994.

[172] W. N. Schilit. *A System Architecture for Context-aware Mobile Computing.* PhD thesis, New York, NY, USA, 1995.

[173] A. Schmidt, M. Beigl, and H. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.

[174] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[175] A. Sherman, P. A. Lisiecki, A. Berkheimer, and J. Wein. ACMS: the akamai configuration management system. In A. Vahdat and D. Wetherall, editors, *2nd Symposium on Networked Systems Design and Implementation (NSDI 2005), May 2-4, 2005, Boston, Massachusetts, USA, Proceedings.* USENIX, 2005.

[176] G. Singh and M. H. Chignell. Components of the visual computer: a review of relevant technologies. *The Visual Computer*, 9(3):115–142, 1992.

[177] T. Sivaharan, G. S. Blair, and G. Coulson. GREEN: A configurable and re-configurable publish-subscribe middleware for pervasive computing. In R. Meersman, Z. Tari, M. Hacid, J. Mylopoulos, B. Pernici, Ö. Babaoglu, H. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*, volume 3760 of *Lecture Notes in Computer Science*, pages 732–749. Springer, 2005.

[178] M. Slater, V. Linakis, M. Usoh, and R. Kooper. Immersion, presence and performance in virtual environments: an experiment with tri-dimensional chess. In M. Green, K. M. Fairchild, and M. Zyda, editors, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST 1996, Hong Kong, July 01-04, 1996*, pages 163–172. ACM, 1996.

[179] S. P. Smith and D. Trenholme. Rapid prototyping a virtual fire drill environment using computer game technology. *Fire safety journal*, 44(4):559–569, 2009.

[180] J. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In J. Bosch, W. M. Gentleman, C. Hofmeister, and J. Kuusela, editors, *Software Architecture:*

System Design, Development and Maintenance, IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture (WICSA3), August 25-30, 2002, Montréal, Québec, Canada, volume 224 of *IFIP Conference Proceedings*, pages 29–43. Kluwer, 2002.

[181] N. A. Streitz, J. Geißler, J. M. Haake, and J. Hol. DOLPHIN: integrated meeting support across local and remote desktop environments and liveboards. In J. B. Smith, F. D. Smith, and T. W. Malone, editors, *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*, pages 345–358. ACM, 1994.

[182] M. R. Stytz. Distributed virtual environments. *IEEE Computer Graphics and Applications*, 16(3):19–31, 1996.

[183] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36, 2010.

[184] I. E. Sutherland. A head-mounted three dimensional display. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '68 Fall Joint Computer Conference, December 9-11, 1968, San Francisco, California, USA - Part I*, volume 33 of *AFIPS Conference Proceedings*, pages 757–764. AFIPS / ACM / Thomson Book Company, Washington D.C., 1968.

[185] S. L. Tanimoto. VIVA: A visual language for image processing. *J. Vis. Lang. Comput.*, 1(2):127–139, 1990.

[186] D. L. Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000.

[187] J. Tumblin and H. E. Rushmeier. Tone reproduction for realistic images. *IEEE Computer Graphics and Applications*, 13(6):42–48, 1993.

[188] U. Varshney. Pervasive healthcare and wireless health monitoring. *Mob. Netw. Appl.*, 12(2-3):113–127, Mar. 2007.

[189] V. Venkatesh and H. Bala. Technology acceptance model 3 and a research agenda on interventions. *Decision Sciences*, 39(2):273–315, 2008.

[190] V. Venkatesh and F. D. Davis. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, 46(2):186–204, 2000.

[191] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis. User acceptance of information technology: Toward a unified view. *MIS Quarterly*, 27(3):425–478, 2003.

[192] P. Verissimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser. Cortex: Towards supporting autonomous and cooperating sentient entities. *Proceedings of European Wireless 2002 (EW2002)*, pages 595–601, 2002.

[193] W3C. Extensible markup language (xml) 1.0 (fifth edition). `https://www.w3.org/TR/REC-xml/`. Online; accessed: August 6, 2019.

[194] R. Want, T. Pering, and D. L. Tennenhouse. Comparing autonomic and proactive computing. *IBM Systems Journal*, 42(1):129–135, 2003.

[195] T. Wark, P. I. Corke, P. Sikka, L. Klingbeil, Y. Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley. Transforming agriculture through pervasive wireless sensor networks. *IEEE Pervasive Computing*, 6(2):50–57, 2007.

[196] T. Weis, M. Handte, M. Knoll, and C. Becker. Customizable pervasive applications. In *4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006), 13-17 March 2006, Pisa, Italy*, pages 239–244. IEEE Computer Society, 2006.

[197] M. Weiser. The computer for the 21 st century. *Scientific american*, 265(3):94–105, 1991.

[198] M. Weiser. Hot topics: ubiquitous computing. *Computer*, 26(10):71–72, Oct 1993.

[199] W. Westera, R. Nadolski, H. G. K. Hummel, and I. G. J. H. Wopereis. Serious games for higher education: a framework for reducing design complexity. *J. Comp. Assisted Learning*, 24(5):420–432, 2008.

[200] L. Williams. Casting curved shadows on curved surfaces. In S. H. Chasen and R. L. Phillips, editors, *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1978, Atlanta, GA, USA, August 23-25, 1978*, pages 270–274. ACM, 1978.

[201] F. Zhu, M. W. Mutka, and L. M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 4(4):81–90, 2005.

Appendix

A. Example applications for BASE and PerFlow

This appendix provides the source code for the example applications used in the evaluation of the implementation effort in Section 8.2. For the BASE and PerFlow applications sending the images the classes starting the main thread and capturing the image are not included, as they are not relevant for the functionality of the middleware. This is also the case for the actual window showing the image to the user in the BASE service and PerFlow receiver application.

A.1. BASE Application

The BASE application is a single class runnable as a thread and captures the screen every two seconds. The screenshot is then sent to all available services in the system.

```java
public class BASEApplication extends Device implements Runnable {
  ServiceRegistry registry;
  ReferenceID ownID = new ReferenceID(SystemID.SYSTEM);

  public BASEApplication() {
    setup(Logging.MAXIMUM_VERBOSITY, 500);

    registry = ServiceRegistry.getInstance();
  }

  @Override public void run() {
    while (true) {
      try {
        Thread.sleep(2000);

        ServiceDescriptor[] descriptors = registry.lookup(new String[]{
            IBASEService.class.getName()}, ServiceRegistry.LOOKUP_BOTH);

        for (ServiceDescriptor d : descriptors) {
          BASEProxy proxy = new BASEProxy();
          proxy.setSourceID(ownID);

          proxy.setTargetID(d.getIdentifier());
          proxy.receiveImage(ScreenCapture.capture());
        }
      } catch (Exception e) {
        Logging.error(getClass(), "Error sending image.", e);
      }
    }
  }
}
```

Listing A.1: BASE example application

A.2. BASE Service

The BASE service for receiving and showing the images is split into four classes and the interface for the service. The main class starts the middleware and registers the service. Further, it needs a BASE proxy, BASE skeleton and the implementation of the actual service.

```java
public class BASEServiceMain extends Device {

    public static void main(String args[]) {
        setup(5000);

        BASEService service = new BASEService();
        BASESkeleton skeleton = new BASESkeleton();
        skeleton.setImplementation(service);

        registerService("BASEService", service, skeleton, new String[]{
            IBASEService.class.getName()});
    }
}
```

Listing A.2: BASE example service: Main class

```java
public interface IBASEService {

    public void receiveImage(SerializableImage img) throws
        InvocationException;
}
```

Listing A.3: BASE example service: Interface

```java
public class BASEService extends Service implements IBASEService {
    ImageViewer viewer = new ImageViewer();

    @Override public void receiveImage(SerializableImage img) throws
        InvocationException {
        viewer.showImage(img.getImage());
    }
}
```

Listing A.4: BASE example service: Service

```java
public class BASEProxy extends Proxy implements IBASEService {

    @Override public void receiveImage(SerializableImage img) throws
        InvocationException {
        Object[] args = new Object[1];
        args[0] = img;
        String method = "void receiveImage(SerializableImage)";
        Invocation invocation = proxyCreateDeferred(method, args);
        proxyInvokeDeferred(invocation);
    }
}
```

Listing A.5: BASE example service: Proxy

```java
public class BASESkeleton extends Skeleton {

    @Override protected Result dispatch(String method, Object[] args) {
        IBASEService impl = (IBASEService) getImplementation();
        try {
            if (method.equals("void receiveImage(SerializableImage)")) {
                impl.receiveImage((SerializableImage) args[0]);
                return new Result(null, null);
            }

            return new Result(null, new InvocationException("Illegal
                signature."));
        } catch (Throwable t) {
            return new Result(null, t);
        }
    }
}
```

Listing A.6: BASE example service: Skeleton

A.3. PerFlow Sender Application

The sender application developed with PerFlow behaves exactly like the BASE application. The only exception beeing, that it does not need to determine the receivers itself.

```java
public class PerFlowApplication extends ConfigurableDevice implements
    Runnable {
    private MessageProxy proxy;

    public PerFlowApplication () {
        setup(Logging.MAXIMUM_VERBOSITY, 500);

        proxy = new MessageProxy();
        proxy.setSourceID(getDeviceID());
        registerConnection(true, true, false, "screenCapture", "
            streaming", IMessage.ABBREVIATION_IMAGE);
    }

    @Override public void run() {
        while (true) {
            try {
                Thread.sleep(2000);

                proxy.sendImage("screenCapture", ScreenCapture.capture())
                    ;
            } catch (Exception e) {
                Logging.error(getClass(), "Error sending image.", e);
            }
        }
    }
}
```

Listing A.7: PerFlow example sender

A.4. PerFlow Receiver Application

In contrast to the BASE service developers do not need to implement interface, proxy, or skeleton for the PerFlow receiver application.

```
public class PerFlowServiceMain extends ConfigurableDevice {

    public static void main(String args[]) {
        setup(5000);

        PerFlowService service = new PerFlowService();

        registerDevice("PerFlowService", service);
        registerConnection("PerFlowService", false, true, false, "
            screenCapture", "streaming", IMessage.ABBREVIATION_IMAGE);
    }
}
```

Listing A.8: PerFlow example receiver: Main class

```java
public class PerFlowService extends AbstractMessageReceiver {
    ImageViewer viewer = new ImageViewer();

    @Override
    public void receiveImage(ReferenceID source, String messageType,
        SerializableImage val) {
        viewer.showImage(val.getImage());
    }
}
```

Listing A.9: PerFlow example receiver: Receiver

B. Reconfiguration times for different configuration sizes

In addition to performance analysis in Section 8.3.1 the following appendix includes two figures with the time needed to reconfigure the system with configuration containing 5 or 20 rules.

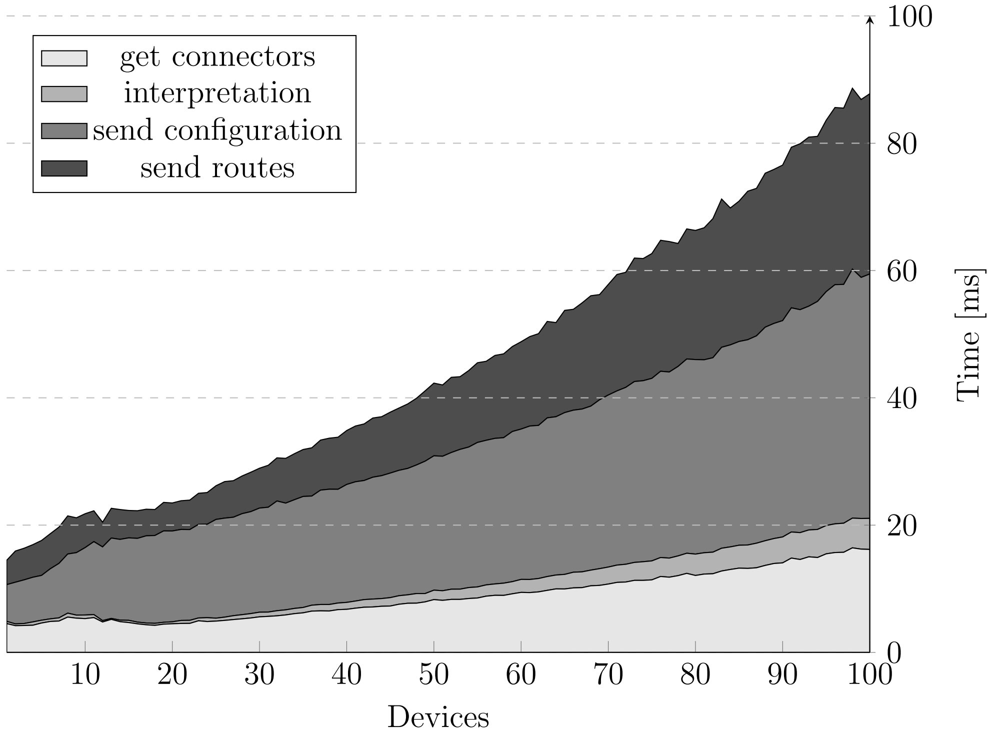


Figure B.1.: Time needed to reconfigure the information flow for up to 100 devices. The values are measured with a configuration containing 5 rules.

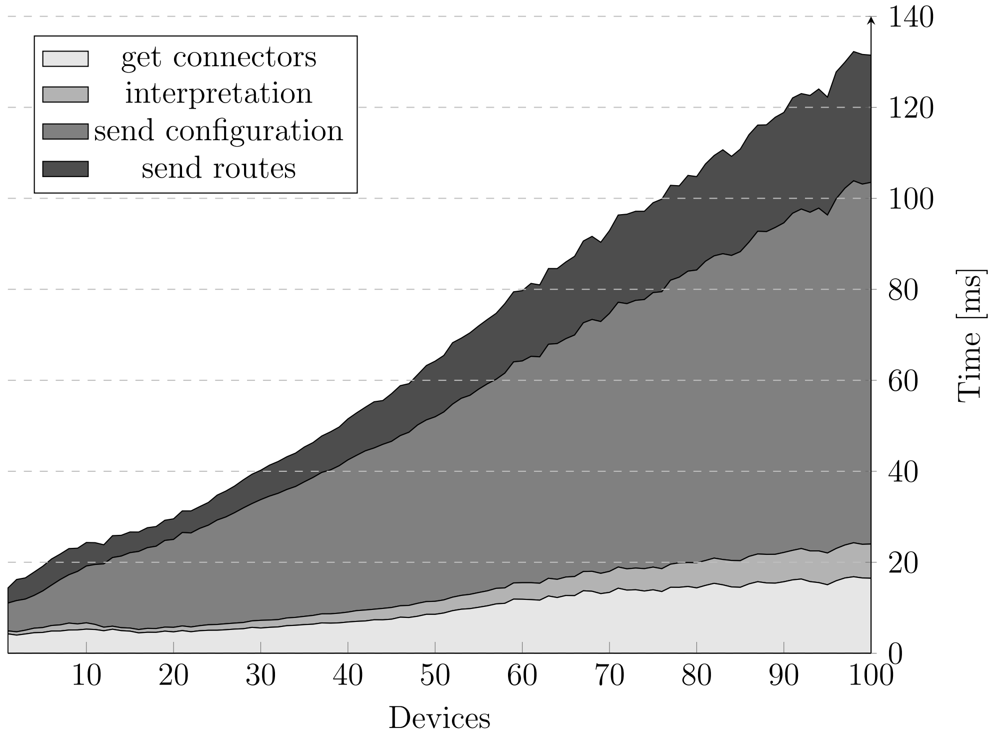


Figure B.2.: Time needed to reconfigure the information flow for up to 100 devices. The values are measured with a configuration containing 20 rules.

C. User Study

The following part of the appendix contains all documents used throughout the two conducted user studies. In Section C.1 the handout for the first user study and in Section C.2 the handout for the second user study is shown. Both contain first the explanation of the PerFlow system and a short example. Afterwards, the scenarios and tasks are explained. Lastly, Section C.3 contains the questionnaire used for both studies.

C.1. PerFlow Tool: First User Study Handout

The handout for the first user study consists of two pages. The first page described the goals of the system and provided a short tutorial on how to use it. Additionally, a small example rule is shown. The second page introduced two scenarios with several tasks each. Overall, the described visual scripting elements and their naming differs from the design introduced in 6.6, as they were refined between the user studies based on the collected feedback.

Visual Scripting Tool- Use Cases

Thank you very much for participating in this scenario/survey which is part of my master thesis about “Visual Scripting for Administrating the Information Flow in Pervasive Systems”.

The goal of the tool is to enable users to control the communication within a smart environment (e.g., Smart Homes) without extensive IT knowledge. Within such systems, we have a variety of different devices and the middleware (MW) is responsible for establishing the network and hereby enables communication between the devices.

The *Visual Scripting Tool* application gets all available devices and message attributes when it is loaded.

The user can then form configurations by formulating rules. *To start a rule, simply drag and drop a node from the list on the left onto the worksheet*. The nodes represent devices or functionality and by wiring nodes together, we can build rules. *To wire nodes together, click on their respective input-/output ports.* A connection between two nodes indicates an information flow from the respective output- to input-port.

When the *Run*-Button is clicked, the application generates a file, which is given to the MW. The MW then, in turn, configures the information flow according to the specified rules. Whenever the middleware receives a message, it checks within the rules where the message shall be sent to.

The tool is applicable to various use cases. We will look at a classroom and an airport use case.

For each of the following scenarios, please complete all tasks by formulating rules using the *Visual Scripting Tool*. After each task, please click the *Run*-Button (!) to save your solution.

To clear the worksheet and remove all nodes and connections, click the *Clear*-Button.

<u>Additional information:</u>

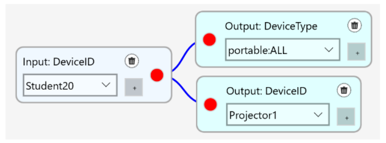
Devices in our Setup have the following attributes:

- DeviceID, a unique name of the owner *e.g., Student17*
- DeviceType, describes the hardware type of the device *e.g., TV*
- UserGroup, defines the group of users that the owner belongs to (usually a role) *e.g., Students*

Messages that are sent within the network consist of the following information:

- MessageID, a unique identifier (not used in our setting)
- MessageType, describes the type of data associated to a message *e.g., Image*
- MessageSource, the deviceID that sent the message (not used in our setting)
- MessageTarget, the target of the message. *e.g., Students:groupA*

A possible rule could look like this:



It means that all messages (of any kind) that are sent from the device “Student20” shall be delivered to all portable devices and additionally to the stationary device “Projector1”. As you can see, an output port can have connections to multiple nodes. Multiple connections to one input port are not allowed, besides the input port of the “FilteringOnMessageType”-Node, so that it can handle the connection to the “PrioritizeMessageType”-Node without having to specify a separate prioritize rule.

To delete a connection, simply click at the port that started the connection. Since connection a node to itself is not allowed, the connector will be deleted.

If you have any questions, feel free to ask me!

Use Case: Smart Classroom

The smart classroom use case illustrates a simple course scenario with 20 students and one instructor. Each person has a personal device in class (IDs: Instructor1, Student1, Student2, Student3, …) and there are two stationary infrastructure devices, Projector1 and Projector2.
In the beginning of the course, the instructor presents the theoretical part of his course. To do so, he sends his presentation to Projector1 because it is bigger and more central to the classroom. Furthermore, he sends a handout to all students that summarizes the content of his presentation. Afterwards, an in-class exercise is on the schedule. The students have to work together in groups to solve different tasks. The instructor determines four groups (group A-D) of five students each and provides different materials to the respective groups. After completing the tasks, the students determine one group leader to present the derived solution. To do so, the leaders send their solution to Projector2. These leaders also send the final solution to the instructor. To prevent a bottleneck at the Instructor- and Projector2-Devices, all messages are sent to the File Server first, and then handed to the target devices.

Tasks:

1. Beginning of the class - Worksheet:
   a. Send message of type Presentation from Instructor1-Device to Projector1-Device.
   b. Send message of type Handout from Instructor1-Device to all devices with user group: Students.
   → Click "Run" and save your configuration. Then, clear the worksheet.
2. In-class exercise - Worksheet:
   a. Determine output device groups: GroupA = Student 1-5, GroupB = Student6-10 (neglect GroupC and GroupD).
   b. Send messages from Instructor1-Device with target group Students:groupA etc. to the respective groups.
   → Click "Run" and save your configuration. Then, clear the worksheet.
3. After group work - Worksheet:
   Send messages from determined group members (GroupA:Student2, GroupB:Student9) to FileServer1 and from FileServer1 to the Instructor1-Device and Projector2-Device.
   → Click "Run" and save your configuration. Then, clear the worksheet.

Use Case: Smart Airport Lounge

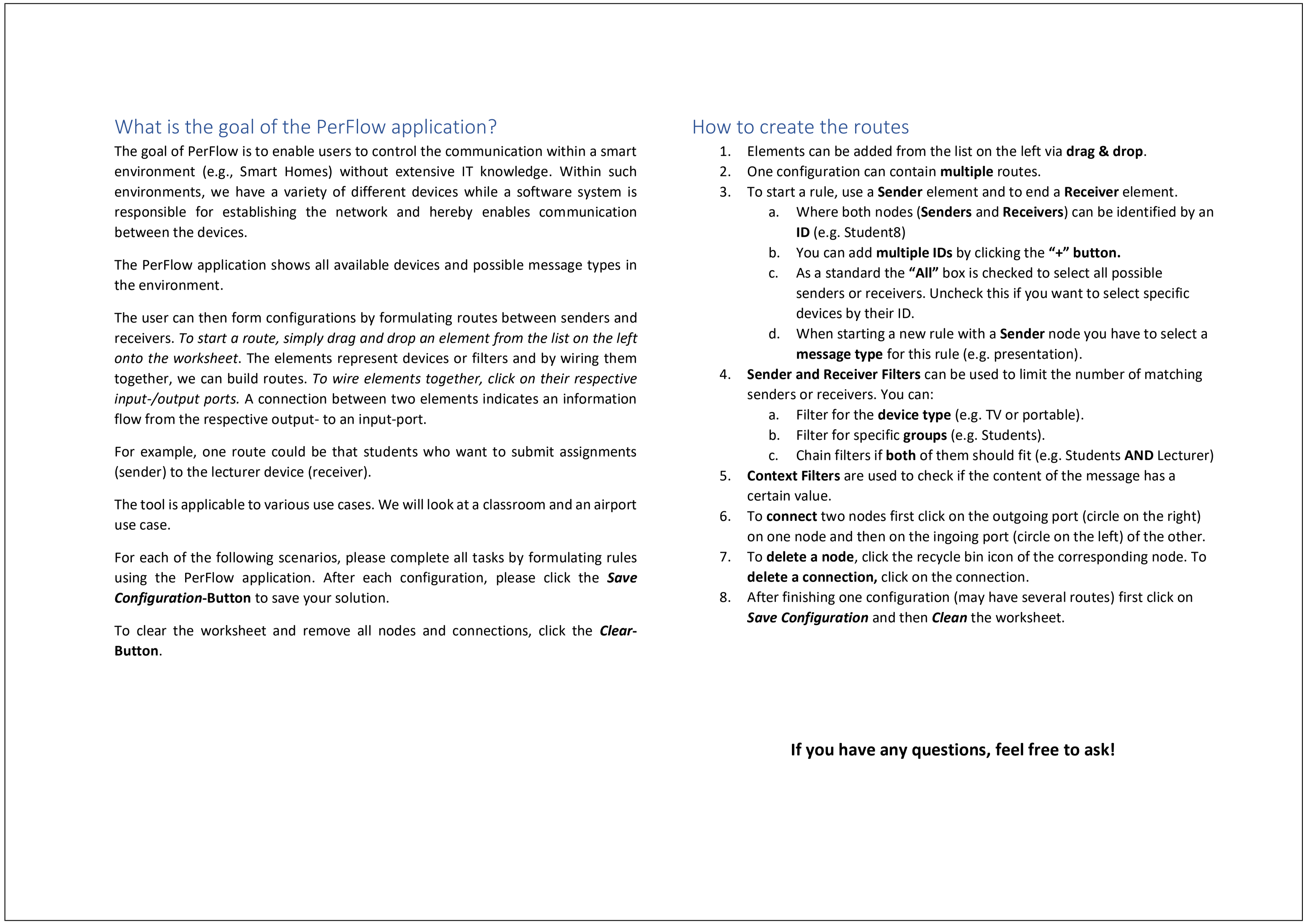
Now, we are in a smart airport lounge. For this use case, we assume that there is a device with ID: CentralSystem that periodically sends messages of different types to stationary and mobile devices within the lounge. The administrator of the middleware can configure this communication by determining what kind of messages are sent to the different device types in the lounge. In order not to miss alarm notifications, these message types are to prioritize.
The devices in this use case are stationary TVs and speaker systems and portable devices such as phones, laptops or tablets from the visitors.
The pervasive system in this use case additionally provides contextual information.
The administrator wants to make use of that. Since the lounge is quite small with only a few visitors at the same time, he decides to give visitors the possibility to watch videos on the stationary TVs if they are close to them and as long as there is no alarm. He decides to specify a range of two meters to determine "closeness".

Configuration – Worksheet:

1. Send videos from CentalSystem to all TVs.
2. Send audio files from CentalSystem to all speaker systems.
3. Send text messages from CentalSystem to all portable devices.
4. Send alarm notifications from CentalSystem to all devices.
5. If the location of any portable device is close (radius: two meters) to the location of any device of type stationary:TV, send messages of type Video to the device of type stationary:TV.
6. Prioritize MessageType ALARM. *(Note: The Prioritize-Node can only be connected to the "FilteringOnMessageType" (FOMT) node and this node can handle multiple incoming connections. So, you don't need to specify a separate rule)*
   → Click "Run" and save your configuration. Then, clear the worksheet.

C.2. PerFlow Tool: Second User Study Handout

The handout for the second user study consists of three pages. The first page describes again the goals of the system and gives a short tutorial on how to use the visual scripting tool. The description is shortened and the tutorial simplified based on the user feedback of the first study. Overall, the wording was less technical. The second page only shows an example route and the third pages introduces the scenarios and tasks.



What is the goal of the PerFlow application?

The goal of PerFlow is to enable users to control the communication within a smart environment (e.g., Smart Homes) without extensive IT knowledge. Within such environments, we have a variety of different devices while a software system is responsible for establishing the network and hereby enables communication between the devices.

The PerFlow application shows all available devices and possible message types in the environment.

The user can then form configurations by formulating routes between senders and receivers. *To start a route, simply drag and drop an element from the list on the left onto the worksheet*. The elements represent devices or filters and by wiring them together, we can build routes. *To wire elements together, click on their respective input-/output ports.* A connection between two elements indicates an information flow from the respective output- to an input-port.

For example, one route could be that students who want to submit assignments (sender) to the lecturer device (receiver).

The tool is applicable to various use cases. We will look at a classroom and an airport use case.

For each of the following scenarios, please complete all tasks by formulating rules using the PerFlow application. After each configuration, please click the ***Save Configuration*-Button** to save your solution.

To clear the worksheet and remove all nodes and connections, click the ***Clear*-Button**.

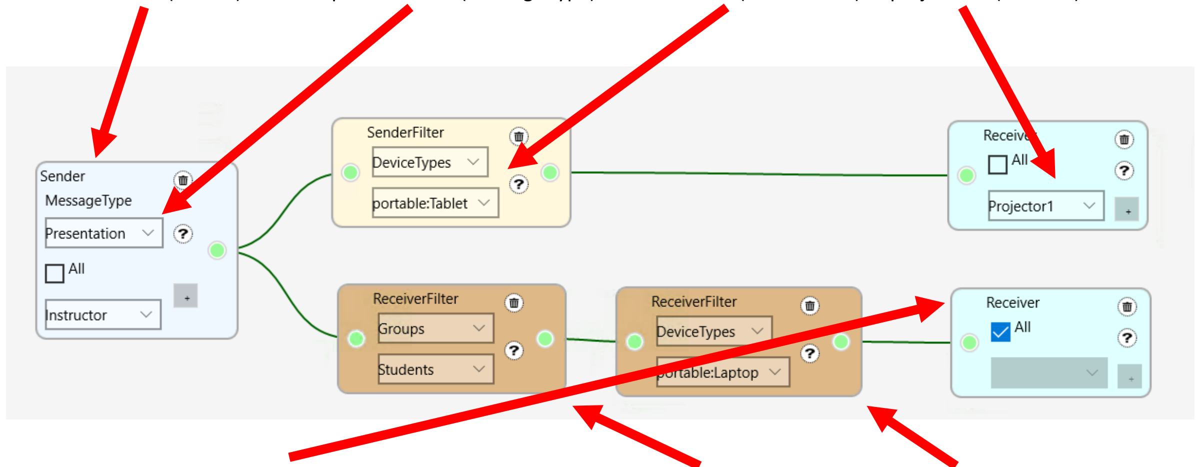
How to create the routes

1. Elements can be added from the list on the left via **drag & drop**.
2. One configuration can contain **multiple** routes.
3. To start a rule, use a **Sender** element and to end a **Receiver** element.
    a. Where both nodes (**Senders** and **Receivers**) can be identified by an **ID** (e.g. Student8)
    b. You can add **multiple IDs** by clicking the **“+” button.**
    c. As a standard the **“All”** box is checked to select all possible senders or receivers. Uncheck this if you want to select specific devices by their ID.
    d. When starting a new rule with a **Sender** node you have to select a **message type** for this rule (e.g. presentation).
4. **Sender and Receiver Filters** can be used to limit the number of matching senders or receivers. You can:
    a. Filter for the **device type** (e.g. TV or portable).
    b. Filter for specific **groups** (e.g. Students).
    c. Chain filters if **both** of them should fit (e.g. Students **AND** Lecturer)
5. **Context Filters** are used to check if the content of the message has a certain value.
6. To **connect** two nodes first click on the outgoing port (circle on the right) on one node and then on the ingoing port (circle on the left) of the other.
7. To **delete a node**, click the recycle bin icon of the corresponding node. To **delete a connection,** click on the connection.
8. After finishing one configuration (may have several routes) first click on ***Save Configuration*** and then ***Clean*** the worksheet.

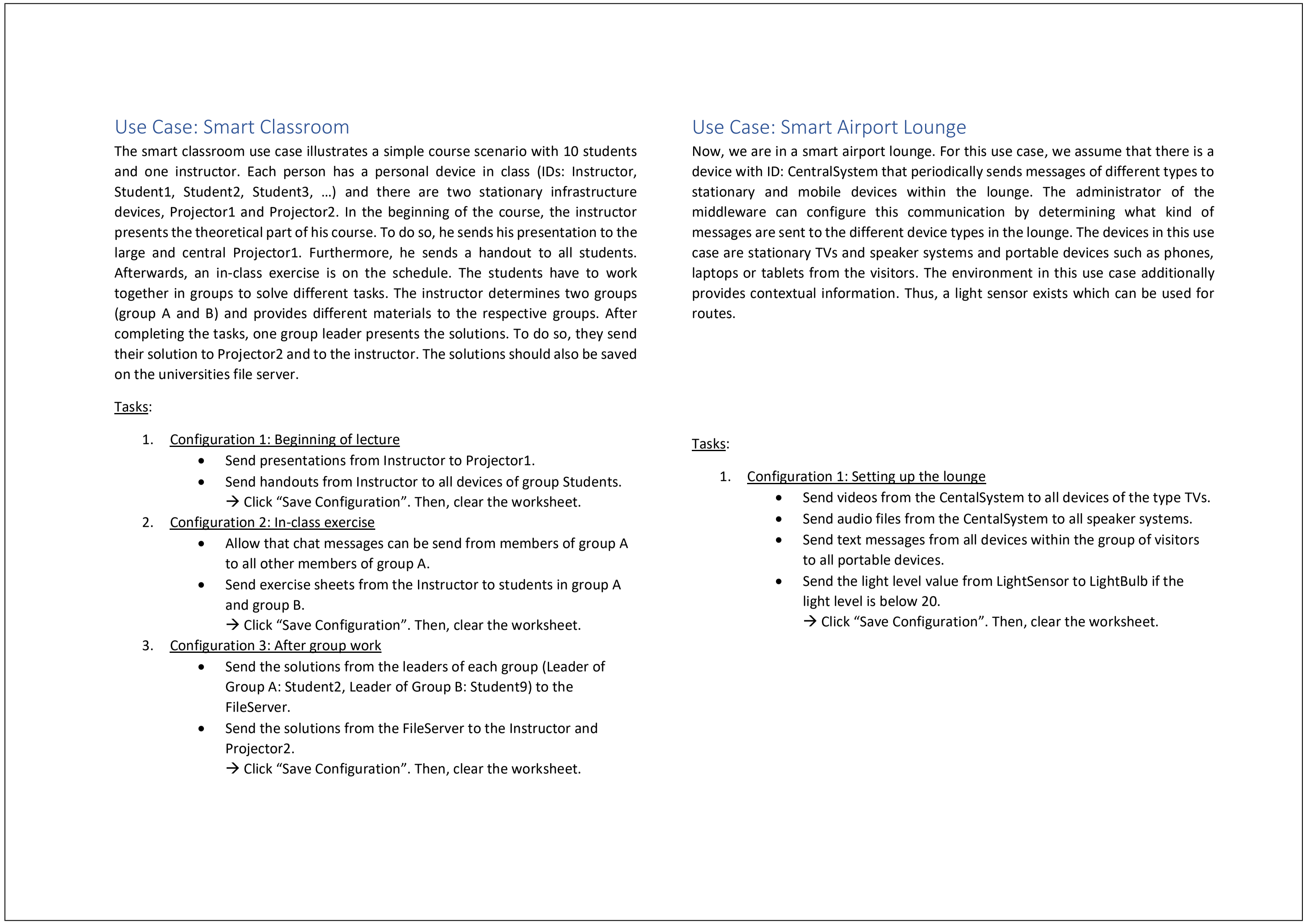
If you have any questions, feel free to ask!

Example

The instructor 1 (sender) can send presentations (message type) from his tablet (sender filter) to projector 1 (receiver).



He can send the presentations to all devices (receiver) which are in the group students and from device type laptop (receiver filters).



Use Case: Smart Classroom

The smart classroom use case illustrates a simple course scenario with 10 students and one instructor. Each person has a personal device in class (IDs: Instructor, Student1, Student2, Student3, …) and there are two stationary infrastructure devices, Projector1 and Projector2. In the beginning of the course, the instructor presents the theoretical part of his course. To do so, he sends his presentation to the large and central Projector1. Furthermore, he sends a handout to all students. Afterwards, an in-class exercise is on the schedule. The students have to work together in groups to solve different tasks. The instructor determines two groups (group A and B) and provides different materials to the respective groups. After completing the tasks, one group leader presents the solutions. To do so, they send their solution to Projector2 and to the instructor. The solutions should also be saved on the universities file server.

Tasks:

1. Configuration 1: Beginning of lecture
    - Send presentations from Instructor to Projector1.
    - Send handouts from Instructor to all devices of group Students.
      → Click “Save Configuration”. Then, clear the worksheet.
2. Configuration 2: In-class exercise
    - Allow that chat messages can be send from members of group A to all other members of group A.
    - Send exercise sheets from the Instructor to students in group A and group B.
      → Click “Save Configuration”. Then, clear the worksheet.
3. Configuration 3: After group work
    - Send the solutions from the leaders of each group (Leader of Group A: Student2, Leader of Group B: Student9) to the FileServer.
    - Send the solutions from the FileServer to the Instructor and Projector2.
      → Click “Save Configuration”. Then, clear the worksheet.

Use Case: Smart Airport Lounge

Now, we are in a smart airport lounge. For this use case, we assume that there is a device with ID: CentralSystem that periodically sends messages of different types to stationary and mobile devices within the lounge. The administrator of the middleware can configure this communication by determining what kind of messages are sent to the different device types in the lounge. The devices in this use case are stationary TVs and speaker systems and portable devices such as phones, laptops or tablets from the visitors. The environment in this use case additionally provides contextual information. Thus, a light sensor exists which can be used for routes.

Tasks:

1. Configuration 1: Setting up the lounge
    - Send videos from the CentalSystem to all devices of the type TVs.
    - Send audio files from the CentalSystem to all speaker systems.
    - Send text messages from all devices within the group of visitors to all portable devices.
    - Send the light level value from LightSensor to LightBulb if the light level is below 20.
      → Click “Save Configuration”. Then, clear the worksheet.

C.3. Questionnaire

Participant Information

What is your gender?

- Female
- Male
- Other

What is your age?

____

Are you working in or studying an IT-related field?

- Yes
- No

Please rate your confidence using a computer or similar technological devices?

- Not at all Confident
- Slightly Confident
- Moderately Confident
- Very Confident
- Extremely Confident

What is your highest educational degree?

____

Scenario-Related Statements

Overall, I am satisfied with the ease of completing the tasks.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Overall, I am satisfied with the amount of time it took to complete the tasks.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Overall, I am satisfied with the provided information for completing the tasks (scenario description).

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Do you have any comments (positive or negative) that are related to the scenarios?

Ease of Use - Statements

I find the application easy to use.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Learning to operate the application was easy for me.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

My interaction with the application was clear and understandable.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

I find the application to be flexible to interact with.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

I find it easy to get the application to do what I want it to do.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Overall, I am satisfied with how easy it is to use this application.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Usefulness - Statements

I find the application useful in completing the tasks (configuring the information flow).

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

I could effectively complete the tasks using this application.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Using the application enables me to accomplish the tasks (more) quickly.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

This application has all the functions and capabilities I expect it to have.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Overall, I find the application useful.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

User Interface and Visual Language - Statements

The organization of information on the screen is clear.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

I find the various functions in this application well integrated.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

The construction of configurations, i.e. rules, is intuitive.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

The functionality of nodes is clear.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

The application gave error messages that clearly told me how to fix problems.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

The application prevented me from doing careless mistakes.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

The on-screen messages provided within this application are clear.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

The application is user friendly.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Overall, I enjoyed using and playing around with the application.

- Strongly Disagree (1)
- ...
- Strongly Agree (7)

Comments

What did you especially like when using the application?

____

What did you not like when using the application?

____

Did you encounter any problems?

____

Further Comments:

____

Thank you for participating in our user study!

C.4. Survey Results

		Study 1	Study 2
Scenario	**Overall, I am satisfied with the ease of completing the tasks.**	5.70	6.00
	Overall, I am satisfied with the amount of time it took to complete the tasks.	5.85	5.96
	Overall, I am satisfied with the provided information for completing the tasks (scenario description).	6.10	5.65
Ease of Use	**I find the application easy to use.**	5.75	6.35
	Learning to operate the application was easy for me.	6.10	6.26
	My interaction with the application was clear and understandable.	5.25	6.17
	I find the application to be flexible to interact with.	5.95	6.09
	I find it easy to get the application to do what I want it to do.	5.80	6.35
	Overall, I am satisfied with how easy it is to use this application.	5.95	6.48
Usefulness	**I find the application useful in completing the tasks (configuring the information flow).**	6.45	6.43
	I could effectively complete the tasks using this application.	6.00	6.13
	Using the application enables me to accomplish the tasks (more) quickly.	5.53	5.83
	This application has all the functions and capabilities I expect it to have.	5.37	5.39
	Overall, I find the application useful.	6.21	6.30
User Interface	**The organization of information on the screen is clear.**	5.95	6.30
	I find the various functions in this application well integrated.	6.11	6.09
	The construction of configurations, i.e. rules, is intuitive.	5.70	6.43
	The functionality of nodes is clear.	5.45	6.26
	The application gave error messages that clearly told me how to fix problems.	5.79	6.06
	The application prevented me from doing careless mistakes.	5.71	5.95
	The on-screen messages provided within this application are clear.	6.20	6.15
	The application is user friendly.	5.80	6.00
	Overall, I enjoyed using and playing around with the application.	6.40	6.26

Table C.1.: The average result for each question and both user studies.

Publications Contained in this Thesis

- J. Naber, D. Schäfer, S. VanSyckel, and C. Becker. Interactive display services for smart environments. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2157-2164. IEEE, 2015.
- J. Naber, C. Krupitzer, and C. Becker. Transferring an interactive display service to the virtual reality. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1-8. IEEE, 2017.
- J. Naber, S. Schmitz, and C. Becker. Perle: A testbed for pervasive middlewares in learning environments. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 474-479. IEEE, 2019.
- J. Naber, M. Pfannemüller, J. Edinger, and C. Becker. Perflow: Configuring the information flow in a pervasive middleware via visual scripting. In *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous*. ACM, 2019. to be puplished.

Lebenslauf

Seit 02/2014	Akademischer Mitarbeiter Lehrstuhl für Wirtschaftsinformatik II
Seit 02/2014	Mitarbeiter IT und Course Material Mannheim Business School
08/2011 – 01/2014	Master of Science Wirtschaftsinformatik Universität Mannheim
08/2008 – 07/2011	Bachelor of Science Wirtschaftsinformatik Universität Mannheim