technologies

Article

Self-Organizing and Self-Explaining Pervasive Environments by Connecting Smart Objects and Applications †

Börge Kordts *, Bennet Gerlach and Andreas Schrader

Institute of Telematics, University of Lübeck, 23562 Lübeck, Germany; bgerlach@itm.uni-luebeck.de (B.G.); schrader@itm.uni-luebeck.de (A.S.)

* Correspondence: kordts@itm.uni-luebeck.de

† This paper is an extended version of our paper published In Proceedings of the 14th ACM International Conference on Pervasive Technologies Related to Assistive Environments (PETRA 2021), Corfu, Greece, 29 June–2 July 2021.

Abstract: In the past decade, pervasive environments have progressed from promising research concepts to available products present in our everyday lives. By connecting multiple smart objects, device ensembles can be formed to assist users in performing tasks. Furthermore, smart objects can be used to control applications, that, in turn, can be used to control other smart objects. As manual configuration is often time-consuming, an automatic connection of the components may present a useful tool, which should take various aspects into account. While dynamically connecting these components allows for solutions tailored to the needs and respective tasks of a user, it obfuscates the handling and ultimately may decrease usability. Self-descriptions have been proposed to overcome this issue for ensembles of smart objects. For a more extensive approach, descriptions of applications in pervasive environments need to be addressed as well. Based on previous research in the context of self-explainability of smart objects, we propose a description language as well as a framework to support self-explaining ambient applications (applications that are used within smart environments). The framework can be used to manually or automatically connect smart objects as well as ambient applications and to realize self-explainability for these interconnected device and application ensembles.

Keywords: smart object guidance; self-reflection; self-organisation; ambient applications; ensembling

check for updates

Citation: Kordts, B.; Gerlach, B.; Schrader, A. Self-Organizing and Self-Explaining Pervasive Environments by Connecting Smart Objects and Applications. *Technologies* **2022**, *10*, 15. https://doi.org/10.3390/technologies10010015

Academic Editor: Fillia Makedon

Received: 27 December 2021
Accepted: 18 January 2022
Published: 24 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Recently, the vision of pervasive computing has been put to practice so that pervasive environments are increasingly enriching our everyday lives. While the smart home is arguably the most prominent and well-received representative of these environments, other smart spaces are advancing fast. In the near future, smart environments will likely assist people in various places, ranging from smart offices to smart hospitals.

Smart environments are characterized by a number of interconnected devices that can be used to extend the abilities of their users and support them in their every day lives. Dynamically configuring the input and output capabilities of smart devices potentially allows for a needs-based provision of interaction techniques tailored to the abilities and preferences of the users. Hence, we propose to decouple input and output of smart devices and to dynamically connect them for specific situations.

Further, decoupling interaction in ambient space from device and application logic has previously been proposed as a way to honor the high level of user mobility, diversity, heterogeneity of context, and changing resources [1], allowing for an even higher level of adaptivity with respect to user abilities and application context. For example, allowing users to bring their own devices or choose out of a list of possible devices in order to connect to other devices and applications in the smart environment can further increase usability and reduce barriers for some users.

However, challenges arise from the heterogeneity of the involved technology, particularly regarding the functionality, and the concept of distributed and connected devices. Manually connecting a multitude of components causes a high configuration effort for the users or providers of such systems, which is exacerbated when decoupling interaction and providing individualized setups. However, following the organic computing paradigm [2], an adaptive connection of ensembles may present a viable approach to reduce this effort. When doing so, while user specifications are vital to keep the user in the loop [3], they sometimes have to be supplemented by expert specifications (particularly regarding safety issues or specific domain knowledge), which can be challenging, especially in adaptive systems.

There are further challenges in realizing the mentioned benefits of decoupling interaction, as not only can the vast interaction possibilities of dynamically connected devices and applications overwhelm users, but the trend towards embedded and increasingly invisible devices even obfuscates their manner of usage altogether (this problem is known as the invisibility dilemma [4]).

While natural interaction techniques were proposed to address these issues, it is argued that Natural User Interfaces are, in fact, not natural [5], for they rely on actions a person would unlikely perform during natural human communication. Instead, the user has to be instructed first, which becomes even more relevant due to the lack of standards considering interaction in smart environments. Towards this end, the automatic generation and provision of instructions may help users to learn how to interact with even complex and dynamic pervasive environments. Research in the field of human–computer interaction underlines the relevance of explanations for the understanding of (and trust in) interactive systems [6–8]. Previous research also points out the relevance of self-explainability as a general feature for next generation digital systems [9] and the need for self-explainability capabilities of intelligent environments [10].

Understandability and (self-)explainability of software systems has been a research topic in different domains, each from its own perspective. For instance, understandability of software systems, their components, and respective (intended) behaviour, with a particular focus on a high-level structural view, plays an important role in software development and maintenance [11,12]. Recently, user interfaces in edge computing-based Internet-of-Things environments have also begun to be taken into account [13]. Other research communities focus on explaining AI decisions (explainable AI) [14–16] or on explaining the system behaviour of self-adaptive systems—some also focusing on smart spaces (cf. Section 3). However, only limited work focuses on explaining interaction options to the users in dynamically composed smart environments.

In our previous work [17–19], we addressed self-explainability and presented a system, called Ambient Reflection, that forms ensembles of smart devices based upon their self-descriptions. We also discussed the need, as well as the related work, for strictly structured self-descriptions of smart devices and introduced the Smart Object Description Language (SODL) to describe capabilities of smart objects [20]. Ambient Reflection directly connects certain input and output capabilities of devices deemed compatible and suitable for the context and the user. As a result, connections are self-organized by the system following the plug-and-play paradigm. Based upon the self-descriptions of the involved devices and the formed connections, Ambient Reflection then generates manuals and tutorials on-the-fly, directly describing every individual realized combination of input from the user and its immediate expected effect.

While this approach is arguably highly adequate for many scenarios, there are some shortcomings. As Ambient Reflection was fundamentally aimed at smart devices with relatively fixed functionality, adding new functionality typically results in modifying or creating new devices in order to do so, even though it could have been realized more practically via an external software component using existing devices. For example, applications can be flexibly developed independently from hardware issues and specific interaction devices. In a concrete scenario, the application can be dynamically coupled with suitable interaction devices. We henceforth refer to such software components, interacting with smart objects in smart environments, as ambient applications.

A further shortcoming of our previous approach becomes prominent when automatically connecting smart objects. Tackling what we termed the smart object matching problem, we realized an algorithm to match devices based on their self-description which focuses on maximizing the number of connections [21]. However, we have found it to be inadequate in scenarios where other aspects are more relevant. Therefore, we present and approach the more general ensembling problem, whose ensembles can also include ambient applications (see Section 5).

Thus, in this paper, we present an extended version of Ambient Reflection, still automatically connecting decoupled input and output (hardware) devices and still generating user manuals and tutorials for their usage on-the-fly, but now focusing on further aspects when selecting connections and also including ambient applications. Notably, the system serves as a proof of concept that supports the implementation of ambient applications that are able to explain their controls and the provided outputs.

We thereby address the following research questions:

1. How can explainability capabilities concerning the interaction be added to systems that are not self-explainable by design?
2. How can ensembles be put together, and in which scenarios are automatically connected ensembles suitable?
3. Can an algorithm be formulated that determines an acceptable solution for connecting (any set of) components in an acceptable amount of time?

Note that this paper is an extended version of our paper published in PETRA 2021 [22].

2. Exemplary Scenario

The following scenario outlines the benefit of dynamically coupled and self-explaining ensembles of smart objects and ambient applications.

Suppose an ambient application shall be used as a tool for an augmentative and alternative communication (AAC) in a care setting where the user resides more or less stationary in a pervasive environment, such as in an Ambient Assisted Living (AAL) setting or in stationary or hospital care as, for instance, it is planned in the research project ACTIVATE [23]. Modern high-tech AAC aids are often realized using hierarchical menus and a touch user interface. To increase accessibility, such a tool can be realized using an application that is controlled by a list of input devices which are selected according to the needs, abilities, and preferences of the respective user. When input and application control is decoupled, a dynamic use of different input devices can be realized based on formalized interfaces.

While suitable devices may be selected by the nursing staff, an automatic selection based on different domain- and user-specific criteria may be preferable.

Decoupling devices and application means that the control of the system must be explained to each user accordingly. In addition, new instruction becomes necessary if another device is to be used. Hence, it makes sense to generate and deliver instructions automatically so that the user is informed about the control. This requires a self-describing capability of the components involved, i.e., the devices, but also the application.

Modern AAC aids include communication boards, but also provide many other features, such as controlling nearby devices, for instance. A more sophisticated assistant can be realized by using an ambient application that enables users to control the ambiance using smart objects. In that case, the control of the smart objects also remains unclear to the users, and they should be instructed accordingly. This becomes even more relevant in care settings where the users reside only temporarily, e.g., hospitals.

3. Related Work

The organic computing paradigm has been proposed to cope with the increasing complexity of modern computer systems, particularly in the context of pervasive environments [24].

The so-called Organic Computing Middleware introduces self-* features to reduce the effort required to manage these distributed computing systems by making use of a Monitor-Analyse-Plan-Execute (MAPE) cycle. Similar to our approach, the middleware reacts to dynamically changing situations using a feedback cycle. However, in contrast to our approach, the middleware is not focused on self-explainability or the provision of usage instructions.

While the composition of general software modules and services for pervasive environments has been studied extensively [25–27], sometimes even with a focus on explaining behaviour [28], the generated explanations are concerned with general system behaviour, not focussing specifically on interaction capabilities. While these can be handled by existing approaches, a clearer focus might improve the quality of the outcome of the composition, the ease of use of the specification process, the efficiency of the system, or the understandability of the generated explanations.

Although being arguably one of the general features of future digital systems [9], self-explainability is a topic that has been scarcely addressed in the field of pervasive computing. Here, we present the most relevant research regarding self-explainability of ambient applications.

Most research in the field of self-explainability of self-adaptive systems is focused on explaining and reasoning why the system has reached its current state regarding the adaptation process.

For instance, explanation capabilities have been addressed based on an explicit representation of history-awareness, allowing humans to reason about the impact that history has on the decision process [29,30]. The authors present an architecture which can be used to record temporal data that can be queried later to explain a system's behaviour and decision making. In this case, the term self-explainability is associated to the capability of a system to answer questions about its past behaviour.

Recently, explainability capabilities of self-adaptive systems in the context of smart environments have been addressed to help users understand the system's behaviour and be able to adequately react when needed [31]. To allow systems to explain why things happen, the authors focus on learning causal models from experiments on the environment as well as observational data.

The focus of the presented research lies in explaining the self-adaption of respective systems. In contrast to our approach, this related work is not specifically targeting explanations regarding the human–computer interaction that includes steps users are required to perform to solve specific tasks bringing them close to their current goal.

In a closely related work, Autexier and Drechsler motivated the need for self-explainability capabilities of intelligent environments with a focus on intelligent assistance processes [10]. They argue that explanations or intrinsic behavioural intuitiveness of assistance processes can improve their acceptance. This becomes particularly relevant for intelligent systems that perform actions which are barely foreseeable and may not be expected by the users.

The authors provided two examples from their own research. The first example addresses automatic driving assistants, such as automatic driving wheelchairs or walkers, that communicate planned movements which are not clearly recognizable by other persons in the room. The authors observed that users were irritated by the movement of these automatic assistants. The second example are multiple smart assistances interacting together and, hence, obscuring the cause of an action. Two processes illustrated are a night surveillance system and a transportation assistance which control doors and lights. The reasons for system actions (open or close doors and switch lights) may stay unclear to the users in some cases. Thus, Autexier and Drechsler suggest the use of explanations for such systems.

Towards this end, they provide design criteria for self-explaining intelligent environments and distinguish between a user level of explanations (a level understandable by non-technically skilled users) and a program level (a programming language or low level specifications of the system behaviour). They further describe specification-defined explanations (reduced explanations containing only a specification relevant for certain actions) and architectural explanations (the modules relevant for an action). The authors argue that partial models, grounded to a specific situation and able to provide an explanation, can be sufficient but must be kept consistent to foster the user's mental models of the system.

In a more recent work, Fey and Drechsler proposed a conceptual framework based on aforementioned layers of explanations [32]. Explanations are modelled as cause–effect relationships that can be combined transitively resulting in a cause–effect chain. They further provide a technical solution for explanations on the functional level and discuss a case study using a robot controller. Their approach requires designers to write additional code used for the reasoning, causing additional cost. Thus, they argue that a designer may choose to implement systems that are only partially self-explaining (e.g., explaining only the most relevant parts) to reduce this cost while still providing self-explainability for critical parts.

While Autexier, Drechsler, and Fey aim at explaining the behaviour of digital systems and intelligent environments to users as well as other persons (indirectly) involved online, interactions between a user and the system play only a subordinate role in their work. Conversely, we focus on describing ambient applications mainly from the interaction perspective, using interaction devices to control ambient applications or ambient applications to control the ambiance (more or less) directly while decoupling input and output. Hence, the description of physical interactions and the corresponding system behaviour becomes more relevant. In contrast to our approach, the conceptual framework of Fey and Drechsler does not provide a concrete description format for such actions as how to perform a certain gesture or the steps required to perform a task.

To summarize, none of the related work presents a system that is capable of both automatically composing interaction devices, smart objects, and ambient applications which respect users as well as expert specifications and further explaining the results of the composition to the user, focussing on the interaction capabilities rather than general system behaviour.

4. A Description Language for Ambient Applications

When providing explanations of systems that are not self-explainable by design, additional information is required, particularly covering different aspects of the interaction, including how to perform certain actions and possibly complex system responses. Self-descriptions of involved components can be used to provide this information. Additionally, these descriptions may be used as a basis for system composition. In our previous work, we introduced a human and machine-readable description language, called the Smart Object Description Language (SODL), that is used to describe components of smart environments.

In order to present the extensions made to the Ambient Reflection framework that enables it to support ambient applications, we first present modifications made to SODL, whose descriptions the framework uses to form connections and generate explanations. For a detailed introduction to the original language, as well as the framework in general, refer to [20].

To gain an intuition, consider ambient applications as (virtual) output devices that can also serve as (virtual) input devices used to control other systems. Due to the possible complexity of arbitrary applications, we use the description language to describe only the control of the applications and possible actions used to control further systems. We describe the interfaces of an application respecting input and output and basic system behaviour, such as selecting menu elements. This means that we consider ambient applications mostly as black boxes and do not model the entire internal system logic, as modelling the overall system behaviour for any possible application in a description language is neither expedient nor useful in practice. However, describing the system behaviour on another level can provide a useful addition to the description of controls and outputs in some cases and should be considered as described by Fey and Drechsler (see also Section 7).

Figure 1 shows example descriptions and indicates connections specifying interactions for a device and application ensemble that reflects the exemplary scenario (described in Section 2). An ambient application that serves as an AAC aid can be controlled using a gesture input device. Besides serving as a communication board, the application can be used to control smart lights. The descriptions for each device and the application consist of three parts, a general part, state groups, and parts of the task analysis describing input and output options, presented in the following sections. A complete version of the descriptions is available online (http://www.ambient.uni-luebeck.de/sodl/sodl-apps/sodl.html, accessed on 17 January 2022).

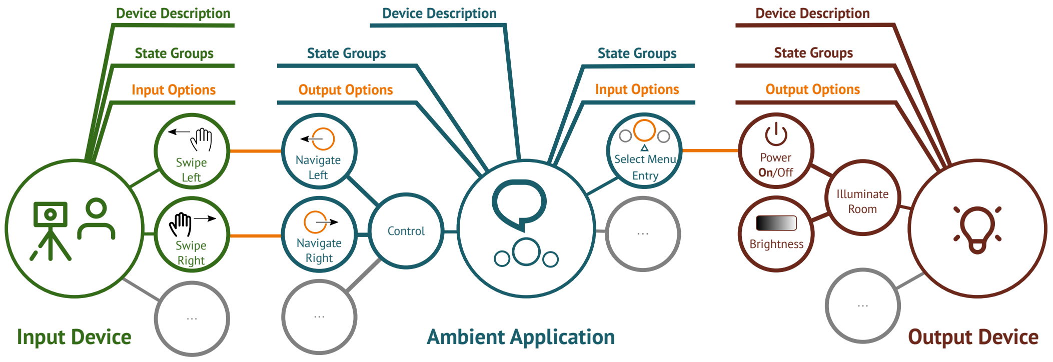


Figure 1. Example descriptions of an input device, an ambient application, and an output device, along with suitable connections. Descriptions consist of device or application details, state groups, as well as input and output options, grouped by tasks. Connections are formed between input and output options, specifying an interaction.

4.1. General Description Section

The device description section of the SODL is used to provide general information about the device based on the product information. It contains information such as manufacturer, model name, product dimensions, etc. The device can be classified according to its spatial use (portable, stationary, or embedded) and the device type can be identified by an ID. Furthermore, a serial number allows for a unique identification of a particular device.

Due to the focus on physical devices, the description did not cover virtual devices or applications. Thus, we extended SODL to also respect virtual devices, particularly when it comes to the spatial use. Thus, the device section becomes a general description section.

4.2. Components, States, and Capabilities

Devices are modelled with their components and respective states. For an easier identification of the devices by the users, images of the device as well as its components can be provided here. Components are distinguished by their type in terms of input and output. A component providing input modalities will be described as an input component. Analogously, components that provide output modalities are recorded as output components.

Additionally, the SODL allows for modelling states of components in state groups (similar to the power states of a lamp) which can contain both discrete (states) and continuous values (states in range). To model interactions and the corresponding system response based on a single state, states can be referenced by an ID. Furthermore, state groups can be addressed (e.g., to toggle states, to transition in a sequence of states, or to address a value from a range).

Of particular relevance is the definition of medial output capabilities to indicate what kind of media can be presented by the component (or device). These capabilities are described using MIME types due to their widespread use for the indication of media types in digital systems. They are used to identify the devices that are capable of presenting manuals as well as tutorials to the users. With this information, Ambient Reflection can decide, on which device(s) the manual will be presented to the user. Ideally, the presentation is mentally and physically as close to the action as possible.

Components, states, and capabilities also apply for ambient applications, while components describe logical components of an application, such as a control component to process inputs. States can be used to model internal states on a coarse level, similar to the states of a navigational control. In principle, more detailed modelling is also possible, but it introduces additional complexity and is only partially supported by the Ambient Reflection framework. Notably, ambient applications potentially provide richer medial output capabilities, particularly when using a graphical interface.

4.3. Task Analysis

The task analysis section of the SODL is used to describe the interaction in greater detail. It is based on two tools: the Hierarchical Task Analysis (HTA) [33] and the Virtual Protocol Model (VPM) [34].

The HTA is based on a task decomposition, a recursive way to split tasks in smaller subtasks until a subtask can be easily solved directly. Tasks are recorded in a tree-structure and the execution order of these tasks is denoted as a plan.

The VPM describes the interaction between human and computer in seven stacked layers. Real-world goals are described at the top layer, the goal layer. These goals can be reached by performing tasks on a computer system (task layer) which consist of operations and the used objects (semantic layer). The syntax layer describes the temporal and spatial order of the input and output operations, whereas the lexical layer is used to depict the smallest information-carrying unit of involved operations and objects. These units consist of atomic commands denoted as Lexemes, which are also known as interaction primitives (alphabetical layer). Finally, the bottom layer, the physical layer, is used to describe physical actions required to communicate.

A decomposition in seven levels of details is also often applied in an HTA and we have previously argued that these two tools can be combined to model interactions on different levels of degrees [19]. While the HTA serves as the underlying structure, it is filled with information about the interactions according to the VPM. Notably, the layer stack can be split into two parts to decouple input and output. One possible way of doing this is separating the description between the alphabetical layer and the lexical layer. While interaction primitives (the smallest addressable element that has a meaningful relation to the interaction itself [35]) are produced by input devices, output devices describe the associated smallest information-carrying unit of involved operations and objects. When mediating input and output devices, interaction primitives can be linked to these units to control the output device and solve objectives.

The task analysis of the SODL begins with the goal layer, which contains a textual description of the goal (e.g., illuminate the room). Any other layer is structured according to the temporal order of execution of its elements (e.g., switch the power state to on, then regulate the brightness). This includes the execution in a sequence or in parallel, but also a choice between possible options on the respective layer. As such an order or choice implicitly describes the syntax of the interaction, the syntactical layer is omitted. While the involved output component and the system response (transition of states) is described on the semantic layer (e.g., power state to on), the involved state group is specified on the lexical layer (e.g., power state). The alphabetic layer carries information on the corresponding input component as well as the respective state group and describes the interaction primitive represented by primitive symbols leading to a state transition in the state group (e.g., swipe left gesture). Finally, physical actions required to perform a gesture, a voice command, etc. are described on the physical layer (e.g., move your hand horizontally to the left).

As an ambient application is not necessarily directly connected to the physical world (e.g., when used as a tool for AAC), in some cases, the exact goal, which describes a real-world concept connected to the world external to the computer, remains unknown to the application. Thus, we allow the omission of the description of a real-world goal and to only describe tasks in these cases (e.g., selecting an element from a menu).

An ambient application controlled by another device can itself be used to control another output device. In this case, the provided input primitives of the application are obviously produced virtually, and there is no description of physical actions for the application. Instead, the physical actions are performed by interacting with the input device used to control the application and are described in the corresponding self-description. Consequently, the physical layer can be omitted for these applications as well. Furthermore, another input type apart from gesture, voice, and touch input is required to describe these virtual inputs. Based on an analysis of typical applications and their user interface elements, we introduced a new input type for virtual inputs that can be used to describe interaction primitives on a virtual level, such as selecting a menu entry, pressing a button, moving a slider, etc. The virtual input can further be linked to a specific control element in the application using an ID. In case none of the given elements is considered suitable, it is possible to describe a custom input type.

We further added the option to link graphical representations to the alphabetical and the semantic layer that can be used to illustrate the input modality as well as the system response. These representations can be used to provide more comprehensive descriptions that also contain images, animations, or videos.

5. The Ensembling Problem

Our description language provides a detailed specification of ambient applications and smart objects from high-level properties or associated tasks down to state groups containing their respective input and output options, thereby specifying possible connections and enabling the formation of highly usable ensembles as well as their self-explanation. However, choosing ensembles that appropriately address the needs of the user is not always straightforward.

The intuitive approach is to have the users themselves choose the connections to be realized for all the ensembles they are going to interact with in their environment. While often adequate, it is not always desirable, as the process of choosing can be overwhelming or tedious in certain situations. Further, the result can be unacceptable, as ideal adaptation to the needs of the user of a pervasive environment can require considerations users are not always able make on-the-fly, such as those related to safety or domain expert knowledge. Furthermore, users are often unaware of the complete functionality of the involved components. However, automatically suggesting suitable ensembles may present a useful tool to assist the user in this process. In the remainder of this section, we define the problem, address related challenges and constraints, present our approach, evaluate its performance and discuss our results.

5.1. Problem Definition

We refer to the problem of suggesting suitable connections between smart objects and applications as the ensembling problem. We thereby naturally extend the related smart object matching problem presented in our previous work, which considered only connections between smart objects, not including ambient applications [21]. While matching mainly focuses on maximizing the number of connections, ensembling is a more general problem that focuses on the overall quality of the ensembles.

We consider the following challenge inherent to the problem: To correctly determine the suitability and to form only appropriate ensembles, all connections for all combinations of applications and smart objects in the environment would have to be evaluated. This, however, presents two potentially difficult tasks. Firstly, the potentially large number of possible connections can introduce high computational complexity. Secondly, sufficiently considering safety or domain expert knowledge can require highly detailed information about the user and the context, which can be difficult to obtain. Furthermore, such knowledge can be hard to specify and, particularly, to formalize in such a way that it can be used by a computer.

The ensembling problem therefore takes on different forms depending on the concrete types of context information that must be considered. Considering only specific types of context information in specific scenarios may allow for the problem to be optimized and reduced so it can be easily solved by specific algorithms. The general case however, depends on possibly arbitrarily complex constraints and criteria. Hence, it cannot be solved efficiently and requires a best effort approach.

5.2. Challenges

Despite the difficulty of the ensembling problem for an arbitrary number of devices, any reasonable algorithm should present an acceptable solution after an acceptable amount of time.

Regarding the acceptable amount of time, we argue that the ensembling happens rarely compared to an interaction with the formed ensembles and, thus, may take longer to be solved. This means the strict real-time constraints for interactions (typically ranging from a few up to a hundred milliseconds) do not apply to this problem. However, computation should typically take, at most, a few minutes, likely even less for scenarios requiring (re)ensembling very frequently (see Section 5.5). In general, for each scenario, different stakeholders including developers, domain experts, administrators, providers (people that provide users access to the ensembles), and users may formulate criteria. The actual devices and some of these criteria may only be known shortly before the usage of the final ensemble. Therefore, the ensembling problem cannot be solved beforehand. Typically, depending on the scenario, either the users or the providers rate the algorithm's responsiveness. To obtain precise constraints on the acceptable amount of time, the specific usage scenario has to be evaluated.

The meaning of the acceptability of the solution obviously depends on the validity of the specified constraints and criteria, including preferences of the user. This is influenced by how well knowledge and preferences can be and actually are formalized, which is reflected by the correlation of the assigned numerical values of the criteria and the perceived goodness of a solution. As is typical in best-effort approaches, solutions are approximated and their acceptability also depends on the numerical difference to the optimal solution. However, acceptable solutions have to respect certain basic constraints. For instance, at least the basic output functions of the devices intended to be controlled must be part of the solution (e.g., switching lights on and off), meaning that there must be a full coverage among mandatory device functions.

5.3. Approach

We previously approached the smart object matching problem based on an algorithm that determines connections between devices. It primarily focuses on finding as many connections as possible (it solves a matching problem) and only takes other criteria into account afterwards. We found this to be unsatisfactory in many scenarios where other criteria are more relevant. Thus, to tackle the ensembling problem, we extend our approach to the smart object matching problem to also work for ambient applications and to allow for more flexibility by taking further aspects into account.

We realized an algorithm that determines a set of connections between smart objects given a set of value functions that each assign a value to the set of connections, each representing a quality criterion. The algorithm is further able to exclude unacceptable solutions via filters representing constraints.

The criteria associated with each value function can be related to very different aspects of ensembles in the environment, ranging from construction details of the devices over usability aspects to expert knowledge. Value functions can be implemented by matching certain value ranges, giving priorities to certain tasks, preferring certain device pairings, etc.

The constraints associated with each filter can exclude solutions due to several different concerns, ranging from safety concerns to provider and user preferences. Filters can be implemented by specifying specific devices or connections to be excluded.

Value functions and filters can be exchanged between runs of the algorithm and the importance of value functions to the solution relative to one another are specified as numerical weights, enabling adjustment to different scenarios.

During each run, the algorithm has to find one of the solutions with the highest numerical value as calculated by the weighted value functions, which also are not excluded by any filter. It does so by first determining all possible connections, then excluding any connections specified by the given filters. Any combination of remaining connections then represents a possible solution and can be evaluated by the combination of weighted value functions.

The number of solutions grows exponentially with the number of possible connections, as each connection can either be part of the ensembles to be realized or not. Thus, a naive brute-force algorithm would take infeasibly long to search through all possible solutions for large numbers of connections. While an algorithmic optimization is conceivable for specific scenarios with fixed value functions, this does not apply to the general case. Given certain combinations of value functions, it is even possible to create very hard to impossible problems.

In order to not depend on specific value functions, but rather tackle the general case, we refrain from employing specific optimization techniques, which could easily be rendered useless in other scenarios using other value functions. Typically, such general problems are tackled by general algorithmic approaches, be they nature-analogous algorithms such as genetic algorithms and neural networks, or probabilistic algorithms [36].

In order to tackle the general ensembling problem and find an acceptable solution in this exponential search space with changing constraints and criteria, we employ a probabilistic brute-force algorithm as a best-effort approach. It samples pseudo-random solutions one after the other for as long as it runs, evaluates them via the combination of value functions and keeps the best one found.

At the core of any probabilistic algorithm lies a randomization scheme, which enables it to sample solutions in parts of the search space that are far apart. While the total amount of solutions searched through is similar to a non-probabilistic brute-force algorithm, this random sampling is widely believed to find better solutions.

We realized the algorithm in Java using a configurable number of threads that sample solutions concurrently beginning at different starting points. We chose a randomization scheme where batches of threads are reset regularly and can earn additional runtime if they find a solution in the vicinity of the sample that improves upon the best solution so far relatively often. Each thread constructs a solution by processing the list of possible connections and randomly selecting or ignoring the next possible connection based on a fixed and individual probability given to the thread and pruning further connections afterwards. Given this random solution, and beginning with the decision for or against the last connection, the thread begins comparing the found solutions with solutions resulting from the inverted decision. Assuming that the last decision was to include the connection, the solution would be compared to a solution that would ignore the connection. When there are connections left to be included or ignored, the thread constructs a new random solution similar to the construction of the starting point but based on the given partial solution. We expect this scheme to perform well when there are clusters of solutions with similar quality, where we believe it is beneficial to continue searching within a cluster until improvements become seldom and to move on to new clusters regularly. However, other randomization schemes, and even other general approaches to the problem, are conceivable. Their respective performance is influenced by the specific scenario and associated constraints and criteria, yet none of them can be expected to perform best in all scenarios.

5.4. Evaluation

While a plethora of constraints and criteria are conceivable, we evaluated the following which we believe to be general, and therefore applicable in various scenarios (for a formal description, see [21]). Obviously, this list is not complete and, depending on the scenario, different criteria or constraints may be important.

- Unambiguousness: An input action should be unambiguously connected to a device function. The input should not be connected to multiple device functions, nor should a device function be controlled by multiple different input actions. This constraint is realized by a filter that only allows one connection per lexeme (element on the lexical layer) or symbol (element on the alphabetic layer).

- Grouping: As previously described, interaction tasks can be grouped into goals. They reflect a combination of tasks and their associated actions required to reach a goal in the physical space. This constraint is realized by a filter that only allows solutions where goals can be reached entirely or are not addressed at all.
- Conformity: Input actions and device functions can each be described by discrete and continuous states. How well their numeric representation suits one another is an important consideration when connecting them. This criterion is realized by considering the difference between the sizes of the states.
- Distance: The physical distance between connected devices influences the usability of an ensemble. Controlling device functions in other rooms or out of sight may obstruct the interaction. This criterion is realized by calculating the distance based on a hierarchical positioning schema.
- Coverage: The understanding of the steering of a device ensemble may be negatively impacted when a large variety of interaction devices has to be used to control a single output device or application. Smart objects should therefore be controlled by a small number of interaction devices. This criterion is realized by calculating the amount of input devices per output.
- Consistency: User experience with existing ensembles should be respected in future ensembles. Previously existing connections between input actions and device functions should therefore be more likely to reoccur. This criterion is realized by preferring solutions containing more already-known connections.

In order to evaluate the probabilistic brute-force algorithm, we utilised Java and constructed two separate JUnit tests to measure its performance in terms of solution goodness and runtime. We implemented the described criteria and weighted them equally. We multiply the value of each criterion to calculate the overall goodness of a solution. In doing so, we filter solutions where a single criterion returns a value of 0. We generated virtual smart objects and virtual ambient applications randomly for each pass and repetition based on fixed parameters. This random generation simulates the heterogeneity of devices and applications conceivable in pervasive environments. The evaluation was conducted on an Ubuntu 18.04 system with a four-core Intel Core i7-7700HQ @ 2.8 GHz and 16 GB DDR4 SDRAM @ 2400 MHz. In each pass, results were logged for further analysis. Each test was performed with the same number of input devices, output devices, and ambient applications. Input and output devices each consisted of up to three components. Ambient applications consisted of up to two input and up to two output components. Each device and application was generated with a random number of state groups (up to five) per component which either consisted of a random number of states (also up to five) for discrete states or a random value range within the bounds of 0 to 100. Furthermore, each generated device or application was assigned a random position in a fixed floor plan. The evaluation data is available online (http://www.ambient.uni-luebeck.de/sodl/sodl-apps/sodl.html, accessed on 17 January 2022).

To measure the performance of our algorithm in terms of the goodness of found solutions, we gradually increased the device and application number for each pass (see Figure 2). Note that, typically, no solution can be found for a small number of devices and applications due to incompatible state groups. While it can be observed that the probabilistic brute force algorithm searches through the entire search space for a small number of devices and applications and therefore finds the optimal solution in these cases, the found solutions diverge increasingly from the optimum for larger numbers. The best solutions are found for about 10 to about 30 devices and applications. However, the algorithm is able to find acceptable solutions even for a large amount of devices and applications. In addition, solutions for the evaluated number of devices exceed a goodness of 0.05 on average.

Due to the exponentially increasing search space, any brute force algorithm could spend almost arbitrary amounts of time searching for solutions. In order to investigate when our algorithm finds acceptable or even good solutions, we measured the average goodness of the solutions found each second while running the algorithm for specific amounts of devices and applications (see Figure 3). The average goodness of the best solution found by our algorithm rapidly increased within the first 15 to 30 s. Afterwards, the goodness increased only gradually. While acceptable solutions were found after about 15 s, the algorithm found good solutions (with respect to the best solution found after about two minutes and even later) after about 30 s. Given this behaviour, we argue that longer runtimes for the algorithm do not provide enough benefit to justify their cost.

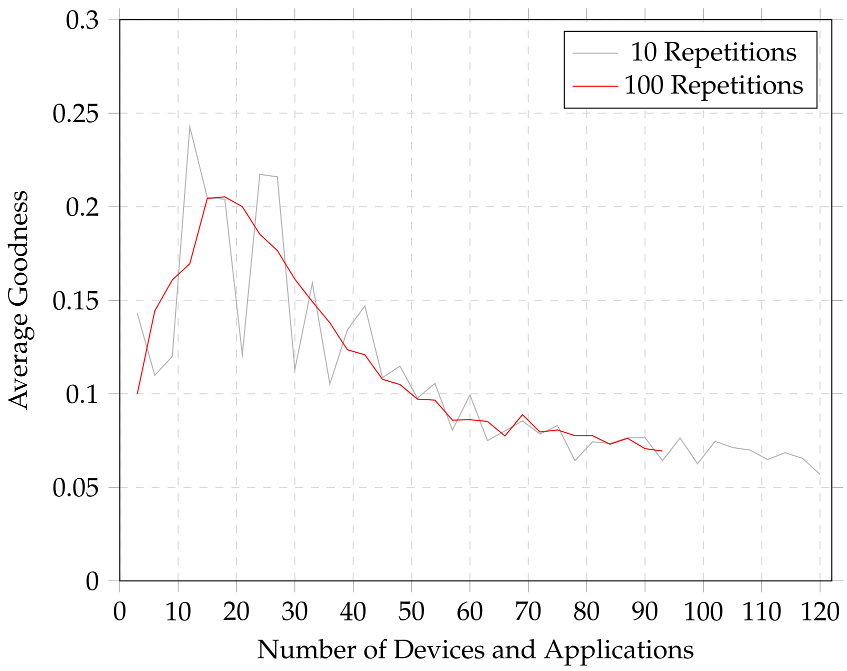


Figure 2. Evaluation results for an increasing number of devices and applications measured on the test system.

5.5. Discussion

In summary, our algorithm meets the requirement of finding solutions within minutes at most. While the average goodness declines with an increasing number of devices and applications, our algorithm is still able to produce solutions of the ensembling problem which seem to asymptotically approximate a positive lower bound. We therefore argue that the algorithm meets the requirement of finding acceptable solutions within an acceptable amount of time, even for scenarios that involve a large number of possible ensembles.

However, the found solutions may be unsatisfactory in some cases. In these cases, additional or more adequate constraints and criteria must be selected to ensure a high quality of the found solutions. These constraints and criteria should be validated to analyse whether the goodness of a solution actually corresponds to a higher acceptance and usability. In addition, with fixed constraints and criteria, an optimized algorithm may be constructed that results in a better performance in terms of both runtime and goodness (possibly even up to optimality). This makes our approach applicable to more scenarios than our previous one for the similar smart object matching problem which focuses on finding as many connections as possible.

While appropriate for many scenarios, the runtime of the algorithm may pose a problem in situations where ensembling is required regularly. For instance, frequent ensembling will likely be required in public spaces where many users interact with the pervasive environment even bringing their own devices. In these cases, further measures can be taken to decrease the runtime while still ensuring a minimal goodness. For instance, instead of computing solutions for all ensembles, an iterative approach may take only the most relevant device functions into account and may incrementally improve upon current ensembles. One way of doing so is to search for connections for mandatory goals first and improve the ensembles in further steps later.

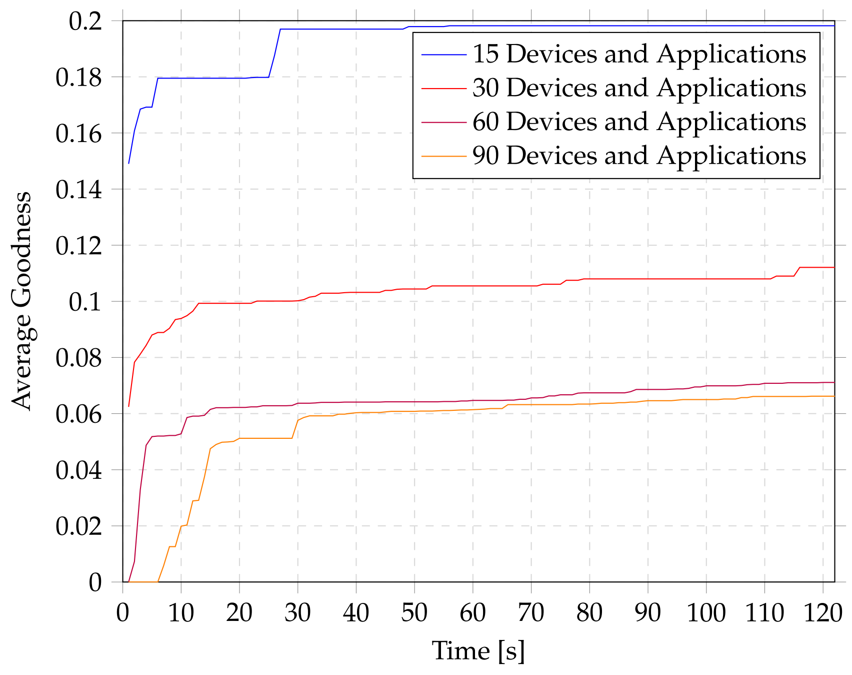


Figure 3. Evaluation results over time for fixed numbers of devices and applications measured on the test system.

6. A Framework for Self-Reflecting Pervasive Environments

In order to provide a framework that allows for self-organizing and self-reflecting pervasive environments based on general ensembles, we developed a system that integrates the aforementioned description language and the proposed approach to the general ensembling problem.

A self-description of ensembles of dynamically connected smart objects and ambient applications can be realized by a system that uses the aforementioned description language for ambient applications and can merge them into an overall description. The Ambient Reflection framework [17] is able to provide such self-descriptions for dynamically coupled device ensembles based on the SODL. Furthermore, the framework is self-organizing by being able to automatically create, restore, and sever device connections and to also replace (parts of) device ensembles to maintain the users' ability to solve their tasks. We extended this to reflect our changes in the SODL and to make it work with ambient applications. This also includes additional requirements that need to be respected.

In essence, the framework consists of two components: devices that make use of the device library, an easy way to integrate devices into the framework, and the description mediator which is used to connect devices based on their functionality, to generate merged descriptions, and to inform the involved devices about both. We provided an additional way to integrate devices or applications in the framework by providing the Virtual Device Daemon (VDD) and corresponding APIs (see Figure 4).

The underlying concept of Ambient Reflection is protocol-agnostic. The reference implementation of Ambient Reflection is realized in Java and provides communication interfaces for the two protocols Devices Profile for Web Services (DPWS) and Universal Plug and Play (UPnP).

The remainder of this section specifies the components of the framework.

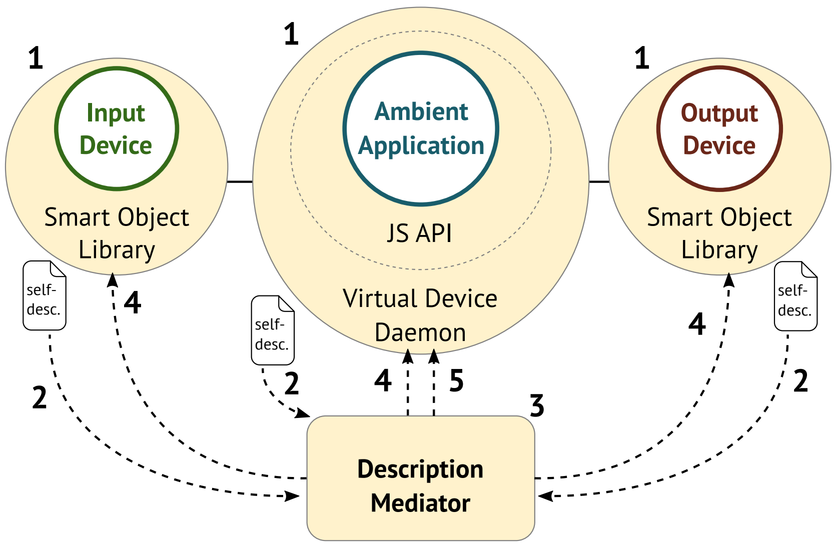


Figure 4. Example connections as realized by several components of the framework: Input and output devices use the Smart Object Library in order to forward their input and output to connected devices and ambient applications. Ambient applications running in the browser employ the Virtual Device Daemon via the Javascript API to do the same. Both devices and applications also provide a self-description to the description mediator, which establishes the connections in the first place. When discovered (1), the description mediator requests for the respective self-descriptions (2). Based on the collected information, the description mediator performs the ensembling and generates instructions (3). The mediator ultimately sends the connection information to the involved components (4) and rendered instructions to components capable of presenting it (5).

6.1. Description Mediator

Both protocols (DPWS and UPnP) provide a discovery process where new devices are discovered in the network and actions can be executed afterwards. This feature is used by Ambient Reflection to query device descriptions based on SODL from the devices. Based on the self-descriptions, interaction primitives and output operations are identified and possible connections are collected. Appropriate inputs and outputs can then be mediated. This can be achieved automatically by using our previously presented algorithm for the ensembling problem (see Section 5). Additionally, a mediation can be specified by a direct configuration of whole ensembles, parts of ensembles, or excluding devices and applications. These configurations can be recorded, for example, in the input or output devices, or by directly configuring the mediation. An overall description of the system can ultimately be created based on the respective device descriptions by merging the descriptions into the overall description (all devices, as well as their respective components, are listed and the complete task analysis consisting of all present layers is assembled). Ambient Reflection can then deliver these merged descriptions to devices capable of displaying the description.

To realize all these features, the so-called description mediator, a component for the mediation, is used (see Figure 5). We integrated the aforementioned changes to the SODL in the description mediator and its mediation process so it is also able to connect ambient applications. Furthermore, we implemented the ensembling algorithm with the presented constraints and criteria and integrated it into the description mediator as well. Consequently, our framework is able to connect devices and applications automatically based upon it. Additionally, we developed a REST interface that can be used to specify configurations for the mediation. Hence, our framework is also capable of controlled self-organization.

The description mediator consists of a communication interface for both protocols, DPWS and UPnP, and an event bus to exchange messages between the mediator's components. After a discovery of devices or applications, the respective meta-information is cached in the Smart Object Information Collector. Additionally, self-descriptions are retrieved and stored there as well. Possible connections are determined in the Fusion Runner and the ensembling is performed by the so-called Matcher which accesses the Fusion Knowledge Base for information about previous solutions and the current connection state. Based on the found solution of the ensembling problem, as well as the respective self-descriptions, an overall description is generated by the device information merger. This description can be rendered for presentation using different rendering engines (see Instruction Factory). Finally, the connection information is delivered via the Connection Handler and the instructions are sent via the Delivery Coordinator. For further explanations, see [19].

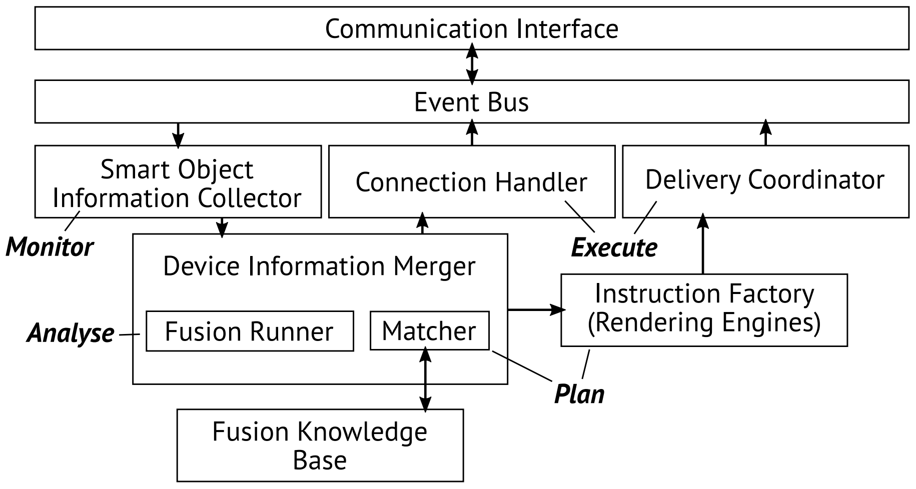
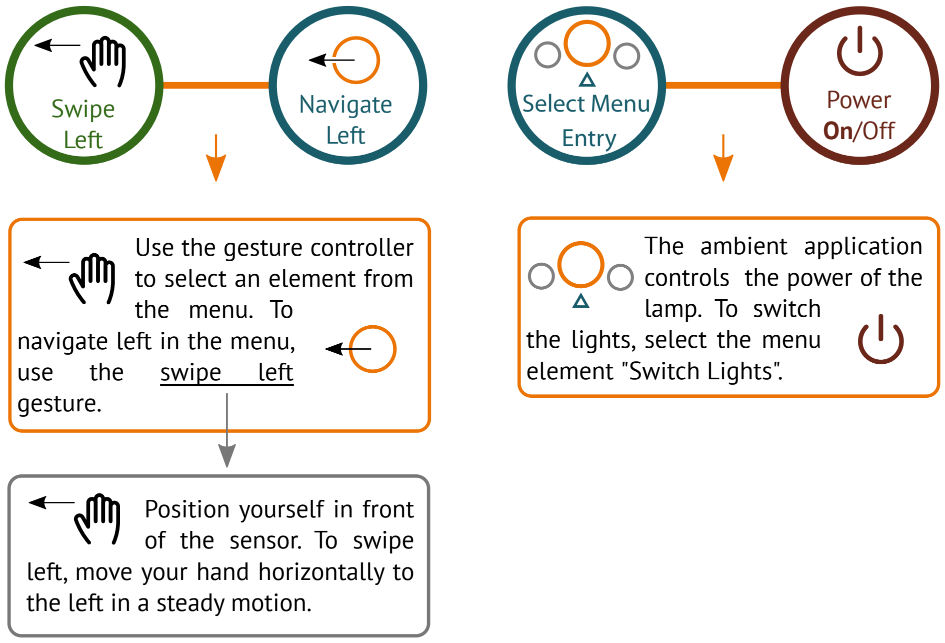


Figure 5. Overview of the description mediator's components.

Figure 6 depicts a small part of the connected inputs and outputs of the example illustrated in Figure 1 (swiping triggers the navigation and selecting the menu entry *Switch Lights* switches the lights) as well as conceivable corresponding descriptions on two levels (cause–effect and physical steps).



Use the gesture controller to select an element from the menu. To navigate left in the menu, use the swipe left gesture.

The ambient application controls the power of the lamp. To switch the lights, select the menu element "Switch Lights".

Position yourself in front of the sensor. To swipe left, move your hand horizontally to the left in a steady motion.

Figure 6. A conceivable manual that describes how to control a dynamically coupled ensemble consisting of an ambient application that can be used to switch lights using a gesture controller as an input device.

Furthermore, we created an HTML rendering engine that respects ambient applications and includes graphical representations. The engine generates static HTML pages that depict how to use the respective interaction technique as well as the system response. The page can be provided by a webserver and presented in a browser. Graphical representations now enrich these tutorials to make the required actions as well as the corresponding system responses easily understandable. Future manuals or tutorial systems can also use the graphical representations to include images, animations, or videos.

6.2. Smart Object Library

The Smart Object Library is a programming interface that can be used to integrate devices into the framework. It is responsible for all network communication required by the framework and is based on an event bus used to process invocations. To create a smart object, the Smart Object Factory, following the factory pattern, can be used. Firstly, the self-description of the smart object is parsed into a data structure that is derived from the SODL using JAXB.

We extended the device library to also process and easily integrate ambient applications in the framework. This includes parsing the description language for ambient applications using JAXB.

While the previous Ambient Reflection framework is only used to exchange messages relating to the interaction, the connection configurations, and the self-descriptions at the various levels, introducing ambient apps to the framework causes additional messages for information unrelated to the relevant interaction. For example, the battery state of wearable input devices can be presented in a graphical interface of an ambient application. Hence, we extended the framework to provide an additional information channel for miscellaneous messages by implementing an additional service. Smart objects register to this service and propagate messages from members of the device and application ensemble to the library. Developers can provide a schema for their messages which can be used to validate incoming messages and finally parse and process the content. For instance, a schema for battery state messages can be used for many devices communicating their battery level to an application. The application may parse those messages and present the state in its interface.

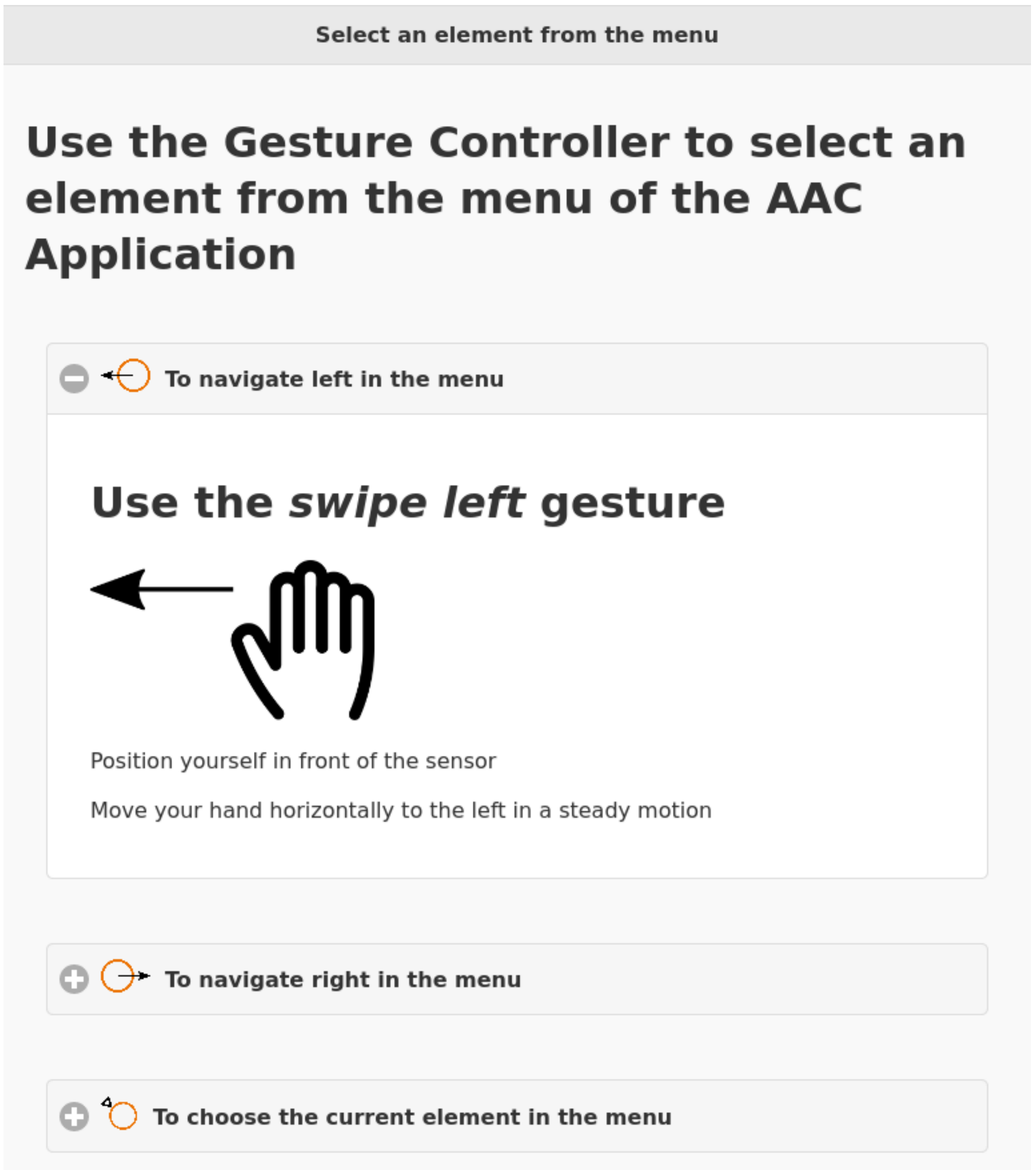
6.3. Virtual Device Daemon and API

In principle, the Smart Object Library can also be used to integrate ambient applications into the framework. To do so, applications have to be implemented in Java and linked to the library. In many cases, however, this is not practical, as applications are usually implemented using other programming languages. In recent years, JavaScript-based applications have been used outside the browser more frequently (e.g., using electron or other tools). Therefore, an interface to the framework, which can be addressed by any other programming language and in particular also from the web browser, can present a great benefit to application developers. Thus, we realized such an interface, called the Virtual Device Daemon (VDD). It is a process that runs in the background and connects to the framework. When executed, the daemon creates a smart object in the sense of the framework based on a provided configuration. Thus, this object is treated by the framework in the same way as any other smart object. It can represent an input or output device, but also an ambient application. All communication with the daemon is routed through web sockets. Therefore, the daemon forms a bridge between the framework and web sockets. As a result, an ambient application or another smart device can be realized by connecting to the daemon.

The Virtual Device API is a collection of different libraries that currently support the programming languages JavaScript, Python, and C#. We have used the libraries for Python and C# primarily for various input devices, while we used the JavaScript API mainly to implement ambient applications. As the framework is based on standardized messages, and those messages are forwarded also based on a standard, the API is easily extensible.

7. Discussion

Our framework can be used to implement a wide range of self-explaining applications controlled by and also controlling dynamically coupled smart objects in assistive environments. For instance, the system can be used to realize the high-tech AAC aid described in our exemplary scenario (see Section 2). By using the framework, an ambient application that supports communication, as well as controlling the ambiance by making use of various smart objects, can be controlled by a list of input devices. The device and application ensemble can be tailored to the user's needs, abilities, and preferences. Instructions of how to use the devices to control the application and how to control devices in the ambiance can be generated by the framework. We provided descriptions for these devices and connected them using our framework. Instructions of the usage were generated by the described HTML rendering engine. A screenshot of a part of these instructions presented in a web browser is shown in Figure 7.



Select an element from the menu

Use the Gesture Controller to select an element from the menu of the AAC Application

To navigate left in the menu

Use the *swipe left* gesture

Position yourself in front of the sensor

Move your hand horizontally to the left in a steady motion

To navigate right in the menu

To choose the current element in the menu

Figure 7. A screenshot of rendered instructions generated by the framework for a gesture device used to control an AAC aid application.

We further used our framework to realize the system in the previously mentioned research project ACTIVATE [37]. By describing input and output devices as well as the application used to support the patient communication, instructions and even interactive tutorials can be generated based on our framework.

Interactive tutorials can be generated, as already previously described [18], but should be adapted to the specifics of ambient applications to ensure high usability. Furthermore, additional information becomes relevant, such as the concrete menu navigation, which can be better addressed through tailored tutorials.

A general description of likely complex application logic using a formal language is, in many cases, neither suitable nor useful; a description of the interactions and their basic resulting actions is often sufficient. Such explanations can be implemented easily at little cost. However, when it comes to systems whose behaviour is not easily foreseeable or expectable, providing more sophisticated explanations should be considered. Though being more expensive, the cost can be reduced by explaining only parts of the system. Particularly for applications that realize basic cause–effect logic, formal descriptions can be implemented with reasonable effort and provide a useful guidance for users of respective applications. Hence, explaining the general system's behaviour should also be considered in certain cases.

While the ensembling functionality of our framework allows it to adapt to the current situation and to minimize configuration overhead, an automatic connection of devices and applications may not always be desirable. On one hand, this targets the question of whether human or machine should be in control and its related implications. On the other hand, there are situations where it is difficult to adequately formulate constraints and criteria that can be used to reliably find viable solutions. This can be attributed to the fact that expert knowledge cannot always be easily formalized accurately by numerical representations; these representations may not provide enough confidence or would be overly complex. For instance, in hospital settings, patients must not be able to control devices or applications of other patients. This constraint can only hardly be realized and thus, semi-automatically forming ensembles by manually choosing the components to connect may pose a pragmatic solution instead. Therefore, automatically connected ensembles can only be suitable in scenarios where constraints and criteria can be adequately formulated with reasonable effort and solutions are accepted by the user without requiring further configuration. Even in other scenarios, however, the capability of our algorithm to find solutions to the ensembling problem may still provide a benefit by reducing the effort required to search for solutions, which could be realized by suggesting several of the best configurations found to the user or provider to choose from. Nevertheless, to fully answer the second part of our second research question of which scenarios automatically connected ensembles are suitable in, further research analysing concrete scenarios is required.

8. Conclusions

In this paper, we presented a framework to realize self-explainability in pervasive environments, including smart objects as well as ambient applications (applications that are used within smart environments). The framework is based on an extension of the Smart Object Description Language (SODL) to support ambient applications. By using this system, smart objects and applications can be connected dynamically to control each other. Furthermore, usage instructions can be generated for these interconnected device and application ensembles. This includes interaction techniques, controls of applications and output devices. For more sophisticated ambient applications, a messaging channel for information unrelated to the interaction, such as battery states, was introduced. This allows applications to present more detailed information about the device and application ensemble as well as to provide further information to the devices. Overall, the presented concept of our system answers our first research question, how explainability capabilities concerning the interaction can be added to systems that are not self-explainable by design, as well as the first part of our second question of how ensembles can be put together.

In the course of realizing the framework, we also addressed the ensembling problem and presented an algorithm capable of finding suitable connections between smart objects and ambient applications, enabling our framework to automatically form ensembles. This answers our third research question of whether an algorithm can be formulated that determines an acceptable solution for connecting (any set of) components in an acceptable amount of time. We also discussed the related second part of the second question, namely, in which scenarios automatically connected ensembles are suitable, based upon our evaluation of the algorithm. In the course of this discussion, we emphasized the importance of formulating adequate ensembling criteria and constraints to the quality of the resulting ensembles. Given our approach, it is now possible to easily integrate formulated criteria reflecting user and expert preferences that may even rely on input from sensors present in the pervasive environment. We also emphasized the need for further research to gain a deeper understanding regarding this research question.

In conclusion, our framework can be used to realize self-explaining and self-organizing smart interaction spaces consisting of smart objects as well as an ambient application for various scenarios with minimal effort. Users may connect compatible devices easily and benefit from the automatic connection process resulting in usable off-the-shelf ensembles and detailed usage instructions. Even future smart objects or applications yet to be developed could simply be connected in a plug-and-play manner. Developers, in turn, may easily realize novel self-describing applications or smart objects using the smart object library as well as the virtual device daemon and its API. They can also develop entire smart interaction spaces based on our framework without worrying about interoperability and compatibility of future components. Hence, we argue that our framework may present a useful tool for developers and users of future smart environments.

9. Future Work

In the following, we describe several plans for future work. An evaluation of the framework that goes beyond a strictly technical analysis is always bound to particular applications and devices. Thus, we plan to evaluate the framework and particularly the generated instructions in different concrete scenarios with suitable ensembles tailored to the target group's needs. In doing so, we plan to investigate the usability of particular instructions but also hope to gather insights on the effectiveness of the overall framework.

We are also working on the provision of multimodal interactive tutorials for ambient applications that can be used to guide a user step-by-step through the use of an application and device ensemble, similar to the tutorials described by Burmeister and Schrader [18]. We further plan to integrate the graphical representations of interaction techniques and the corresponding system response to provide more elaborate tutorials using images, animations, and videos.

Such graphical representations may also cover motion capture data that preserve physical actions required to provide inputs (e.g., gestures or choreographies) that could be used to animate the required actions in a tutorial. Towards this end, using a formal motion description language to describe the required physical actions could further enhance self-descriptions and tutorials. That way, descriptions would stay both human- and machine-readable, while allowing tutorials to generate detailed animations of the expected user's actions.

Additionally, we are planning to develop and test several ambient applications we consider useful, some of which even push the boundaries of our definition of ambient application. A first, very special example is an ambient application to deliver the generated instructions for all ensembles in the environment, showing what is possible in the environment. Having an app for this purpose enables the user to configure the ensembles, for instance deciding which output devices to provide explanations to and which to use in the ensembles themselves. A second app is an ambient app store with functionality similar to mobile app stores, enabling users to download new apps and manage the set of those currently active. A third app is an ambient space design studio, enabling users or providers to configure the connections to be realized in the environment comparable to the approach of Goumopoulos and Mavrommati [38]. They can do so manually or by choosing from suggestions given by our algorithm and even configuring the criteria and constraints for the algorithm that the suggestions are based on. This, in conjunction with the variety of criteria and constraints able to be formulated within our framework, may allow for even better adaptation to the scenario. We are also investigating further universally useful system apps, even inspired by traditional operating systems.

A further category of non-standard ambient applications we are investigating are device-enhancing apps. These apps are not controlled as other apps are, but rather represent a software component, that enables new functionality when used in conjunction with existing devices in an ensemble. Examples are providing gesture recognition for simple cameras, text-to-speech for simple speakers, or timer functionality to simple remote-controlled light bulbs. For many of these apps, we are currently investigating their explainability, as they can be used to provide rather complex interactions.

Other application scenarios in pervasive environments are also conceivable. Our framework can be used to provide self-explainability, manuals, or tutorials for interaction in dynamically changing settings in smart homes, smart offices, smart hospitals, or other smart spaces.

Author Contributions: Conceptualization, B.K., B.G. and A.S.; methodology, B.K. and B.G.; software, B.K. and B.G.; validation, B.K. and B.G.; writing—original draft preparation, B.K., B.G. and A.S.; writing—review and editing, B.K., B.G. and A.S.; visualization, B.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The evaluation data is available online (http://www.ambient.uni-luebeck.de/sodl/sodl-apps/sodl.html, accessed on 17 January 2022).

Acknowledgments: We thank numerous freeicons (https://freeicons.io, accessed on 17 January 2022) members for making their icons available through creative commons rights (CC BY 3.0, https://creativecommons.org, accessed on 17 January 2022): Soni Sokell (tripod icon), Raj Dev (power and user icons), Gayrat Muminov (lamp icon) and Free Preloaders (chat bubble icon). We further thank Wishforge Games (hand icon) for making their icon available for free.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Altakrouri, B. Ambient Assisted Living with Dynamic Interaction Ensembles. Ph.D. Thesis, Universität zu Lübeck, Lübeck, Germany, 2014.
2. Müller-Schloer, C. Organic Computing– on the Feasibility of Controlled Emergence. In Proceedings of the International Conference on Hardware/Software Codesign and Systems Synthesis, Stockholm, Sweden, 8–10 September 2004; IEEE Computer Society: Los Alamitos, CA, USA , 2004; pp. 2–5. [CrossRef]
3. Streitz, N.A.; Rocker, C.; Prante, T.; van Alphen, D.; Stenzel, R.; Magerkurth, C. Designing Smart Artifacts for Smart Environments. *Computer* **2005**, *38*, 41–49. [CrossRef]
4. Kranz, M.; Holleis, P.; Schmidt, A. Embedded Interaction: Interacting with the Internet of Things. *IEEE Internet Comput.* **2010**, *14*, 46–53. [CrossRef]
5. Norman, D.A. Natural User Interfaces Are Not Natural. *Interactions* **2010**, *17*, 6–10. [CrossRef]
6. Kulesza, T.; Stumpf, S.; Burnett, M.; Yang, S.; Kwan, I.; Wong, W. Too Much, Too Little, or Just Right? Ways Explanations Impact End Users' Mental Models. In Proceedings of the 2013 IEEE Symposium on Visual Languages and Human Centric Computing, San Jose, CA, USA, 15–19 September 2013; pp. 3–10. [CrossRef]
7. Dey, A.K. Explanations in Context-Aware Systems. In Proceedings of the Fourth International Conference on Explanation-Aware Computing (ExaCt), Pasadena, CA, USA, 11–12 July 2009; AAAI Press: Palo Alto, CA, USA, 2009; pp. 84–93.
8. Lim, B.Y.; Dey, A.K.; Avrahami, D. Why and Why Not: Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. In Proceedings of the CHI '09: SIGCHI Conference on Human Factors in Computing Systems, Boston, MA, USA, 4–9 April 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 2119–2128. [CrossRef]
9. Drechsler, R.; Lüth, C.; Fey, G.; Güneysu, T. Towards Self-Explaining Digital Systems: A Design Methodology for the Next Generation. In Proceedings of the 2018 IEEE 3rd International Verification and Security Workshop (IVSW), Costa Brava, Spain, 2–4 July 2018; pp. 1–6. [CrossRef]

10. Autexier, S.; Drechsler, R. Towards Self-Explaining Intelligent Environments. In Proceedings of the 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 29–31 August 2018; pp. 1–6.
11. Akour, M.; Aldiabat, S.; Alsghaier, H.; Alkhateeb, K.; Alenezi, M. Software architecture understandability of open source applications. *Int. J. Comput. Sci. Inf. Secur.* **2016**, *14*, 65.
12. Link, D.; Behnamghader, P.; Moazeni, R.; Boehm, B. The Value of Software Architecture Recovery for Maintenance. In Proceedings of the ISEC'19: 12th Innovations on Software Engineering Conference (Formerly Known as India Software Engineering Conference), Pune, India, 14–16 February 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 1–10. [CrossRef]
13. Seo, Y.S.; Huh, J.H. GUI-based Software Modularization through Module Clustering in Edge Computing Based IoT Environments. *J. Ambient. Intell. Humaniz. Comput.* **2019**, 1–15. [CrossRef]
14. Samek, W.; Montavon, G.; Vedaldi, A.; Hansen, L.K.; Müller, K.R. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*; Springer Nature Switzerland AG: Cham, Switzerland, 2019; Volume 11700.
15. Goebel, R.; Chander, A.; Holzinger, K.; Lecue, F.; Akata, Z.; Stumpf, S.; Kieseberg, P.; Holzinger, A. Explainable AI: The New 42? In *Machine Learning and Knowledge Extraction*; Holzinger, A., Kieseberg, P., Tjoa, A.M., Weippl, E., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2018; pp. 295–303. [CrossRef]
16. Holzinger, A. From Machine Learning to Explainable AI. In Proceedings of the 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA), Košice, Slovakia, 23–25 August 2018; pp. 55–66. [CrossRef]
17. Burmeister, D.; Altakrouri, B.; Schrader, A. Ambient Reflection: Towards Self-Explaining Devices. In Proceedings of the 1st Workshop on Large-Scale and Model-Based Interactive Systems: Approaches and Challenges, LMIS 2015, Co-Located with 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2015), Duisburg, Germany, 23–26 June 2015; ACM: Duisburg, Germany, 2015; pp. 16–20.
18. Burmeister, D.; Schrader, A. Runtime Generation and Delivery of Guidance for Smart Object Ensembles. In *Advances in Neuroergonomics and Cognitive Engineering*; Ayaz, H., Mazur, L., Eds.; Advances in Intelligent Systems and Computing; Springer International Publishing: Cham, Switzerland, 2018; pp. 287–296. [CrossRef]
19. Burmeister, D. Selbstreflexive Geräteverbünde in Smarten Umgebungen. Ph.D. Thesis, Universität zu Lübeck, Lübeck, Germany, 2018.
20. Burmeister, D.; Burmann, F.; Schrader, A. The Smart Object Description Language: Modeling Interaction Capabilities for Self-Reflection. In Proceedings of the 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, USA, 13–17 March 2017; pp. 503–508. [CrossRef]
21. Burmeister, D.; Gerlach, B.; Schrader, A. Formal Definition of the Smart Object Matching Problem. *Procedia Comput. Sci.* **2018**, *130*, 302–309. [CrossRef]
22. Kordts, B.; Gerlach, B.; Schrader, A. Towards Self-Explaining Ambient Applications. In Proceedings of the 14th ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA 2021), Corfu, Greece, 29 June–2 July 2021; ACM: New York, NY, USA, 2021. [CrossRef]
23. Kordts, B.; Kopetz, J.P.; Balzer, K.; Jochems, N. Requirements for a System Supporting Patient Communication in Intensive Care in Germany. In *Zukunft Der Pflege Tagungsband Der 1. Clusterkonferenz 2018*; BIS-Verlag der Carl von Ossietzky Universität Oldenburg: Oldenburg, Germany, 2018.
24. Roth, M.; Schmitt, J.; Kiefhaber, R.; Kluge, F.; Ungerer, T. Organic Computing Middleware for Ubiquitous Environments. In *Organic Computing—A Paradigm Shift for Complex Systems*; Springer: Basel, Switzerland, 2011; pp. 339–351. [CrossRef]
25. Bellavista, P.; Corradi, A.; Foschini, L.; Monti, S. Improved Adaptation and Survivability via Dynamic Service Composition of Ubiquitous Computing Middleware. *IEEE Access* **2018**, *6*, 33604–33620. [CrossRef]
26. Delcourt, K.; Adreit, F.; Arcangeli, J.P.; Hacid, K.; Trouilhet, S.; Younes, W. Automatic and Intelligent Composition of Pervasive Applications–Demonstration. In Proceedings of the 19th IEEE International Conference on Pervasive Computing and Communications (PerCom 2021), Kassel, Germany, 22–26 March 2021.
27. Koussaifi, M.; Trouilhet, S.; Arcangeli, J.P.; Bruel, J.M. Ambient Intelligence Users in the Loop: Towards a Model-Driven Approach. In *Software Technologies: Applications and Foundations*; Mazzara, M., Ober, I., Salaün, G., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2018; pp. 558–572. [CrossRef]
28. Koussaifi, M.; Trouilhet, S.; Arcangeli, J.P.; Bruel, J.M. Automated User-Oriented Description of Emerging Composite Ambient Applications. In Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering (SEKE 2019), Lisbon, Portugal, 10–12 July 2019; pp. 473–478.
29. Garcia Dominguez, A.; Bencomo, N.; Parra Ullauri, J.M.; Garcia Paucar, L.H. Towards History-Aware Self-Adaptation with Explanation Capabilities. In Proceedings of the 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W), Umea, Sweden, 16–20 June 2019; pp. 18–23. [CrossRef]
30. Parra-Ullauri, J.M.; García-Domínguez, A.; García-Paucar, L.H.; Bencomo, N. Temporal Models for History-Aware Explainability. In Proceedings of the SAM'20: 12th System Analysis and Modelling Conference, Montreal, QC, Canada, 19–20 October 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 155–164. [CrossRef]
31. Fadiga, K.; Houzé, E.; Diaconescu, A.; Dessalles, J.L. To Do or Not to Do: Finding Causal Relations in Smart Homes. In Proceedings of the 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), Washington, DC, USA, 27 September–1 October 2021; IEEE: Washington, DC, USA, 2021. [CrossRef]

32. Fey, G.; Drechsler, R. Self-Explaining Digital Systems: Technical View, Implementation Aspects, and Completeness. In *Advanced Boolean Techniques: Selected Papers from the 13th International Workshop on Boolean Problems*; Drechsler, R., Soeken, M., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 1–20.
33. Stanton, N.A. Hierarchical Task Analysis: Developments, Applications, and Extensions. *Appl. Ergon.* **2006**, *37*, 55–79. [CrossRef] [PubMed]
34. Nielsen, J. A Virtual Protocol Model for Computer-Human Interaction. *Int. J.-Man-Mach. Stud.* **1986**, *24*, 301–312. [CrossRef]
35. Niezen, G. Ontologies for Interaction: Enabling Serendipitous Interoperability in Smart Environments. Doctoral Thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2012.
36. Mitzenmacher, M.; Upfal, E. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*; Cambridge University Press: Cambridge, UK, 2017.
37. Kordts, B.; Kopetz, J.P.; Schrader, A. A Framework for Self-Explaining Systems in the Context of Intensive Care. In Proceedings of the 2021 IEEE International Conference on Autonomic Computing and Self- Organizing Systems (ACSOS), Washington, DC, USA, 27 September–1 October 2021; IEEE: Washington, DC, USA, 2021. [CrossRef]
38. Goumopoulos, C.; Mavrommati, I. A Framework for Pervasive Computing Applications Based on Smart Objects and End User Development. *J. Syst. Softw.* **2020**, *162*, 110496. [CrossRef]