Towards Self-Explaining Ambient Applications

Börge Kordts
Institute of Telematics
University of Lübeck
Lübeck, Germany
kordts@itm.uni-luebeck.de

Bennet Gerlach
Institute of Telematics
University of Lübeck
Lübeck, Germany
bgerlach@itm.uni-luebeck.de

Andreas Schrader
Institute of Telematics
University of Lübeck
Lübeck, Germany
schrader@itm.uni-luebeck.de

ABSTRACT

In the past decade, pervasive environments have emerged 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. 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 dynamically connect smart objects as well as ambient applications and to realize self-explainability for these interconnected device and application ensembles.

CCS CONCEPTS

• **Computer systems organization** → *Self-organizing autonomic computing*; • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Applied computing** → **Computer-assisted instruction**.

KEYWORDS

smart object guidance, self-reflection, ambient applications

ACM Reference Format:
Börge Kordts, Bennet Gerlach, and Andreas Schrader. 2021. Towards Self-Explaining Ambient Applications. In *The 14th PErvasive Technologies Related to Assistive Environments Conference (PETRA 2021), June 29-July 2, 2021, Corfu, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3453892.3461325

1 INTRODUCTION

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. Decoupling input and output of smart devices by dynamically connecting them allows for a needs-based provision of interaction techniques tailored to the abilities and preferences of the users.

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.

There are however challenges in realizing these benefits, 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 [12]).

While natural interaction techniques were proposed to address these issues, it is argued that Natural User Interfaces are in fact not natural [17] 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 [8, 13, 14]. 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 [2].

In our previous work [3–7], we addressed self-explainability and presented a system, called *Ambient Reflection*, that forms *ensembles* of smart devices based upon their self-descriptions. It directly connects certain input and output capabilities of devices deemed compatible and suitable for the context and the user. 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. Since 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

PETRA 2021, June 29-July 2, 2021, Corfu, Greece
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8792-7/21/06.
https://doi.org/10.1145/3453892.3461325

been realized more practically via an external software component using existing devices. We henceforth refer to such software components, interacting with smart objects in smart environments, as *ambient applications*.

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, based on self-descriptions of the respective components, but now 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 self-explainability of device and application ensembles in pervasive environments, focusing on interaction possibilities and possible controls of applications as well as various output devices.

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, like in an AAL setting or in stationary or hospital care, as for instance it is planned in the research project ACTIVATE [11]. 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.

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, like 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, also the control of the smart objects 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, like in hospitals, for instance.

3 RELATED WORK

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.

Autexier and Drechsler motivate the need for self-explainability capabilities of intelligent environments with a focus on intelligent assistance processes [2]. 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 provide two examples from their own research. The first example addresses automatic driving assistants, like 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 allowing 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 propose a conceptual framework based on aforementioned layers of explanations [10]. Explanations are modeled 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. Opposingly, 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, like how to perform a certain gesture or the steps required to perform a task.

4 A DESCRIPTION LANGUAGE FOR AMBIENT APPLICATIONS

In order to present the extensions made to the Ambient Reflection framework that enables it to support ambient applications, we first present the extensions made to the human and machine-readable description language, called the Smart Object Description Language (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 [5].

To get an intuition, consider ambient applications as (virtual) output devices that can also serve as (virtual) input devices for 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 devices. We describe the interfaces of an application respecting input and output and basic system behaviour, like 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. Yet, 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 6).

Figure 1 shows example descriptions and indicates connections specifying interactions for a device and application ensemble that reflects the examplary 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.

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 like manufacturer, model name, product dimensions, etc. The device can be classified according to its spatial use (like 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 modeled 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.

Additionally, the SODL allows for modeling states of components in state groups (like 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 as close to the action as possible.

Components, states, and capabilities also apply for ambient applications, while components describe logical components of an application, like a control component to process inputs. States can be used to model internal states on a coarse level, like the states of a navigational control. In principle, more detailed modeling 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) [18] and the Virtual Protocol Model (VPM) [15].

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). The bottom layer, the physical layer, is finally 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 [3]. 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 [16]) are produced by input devices, output devices describe the associated smallest

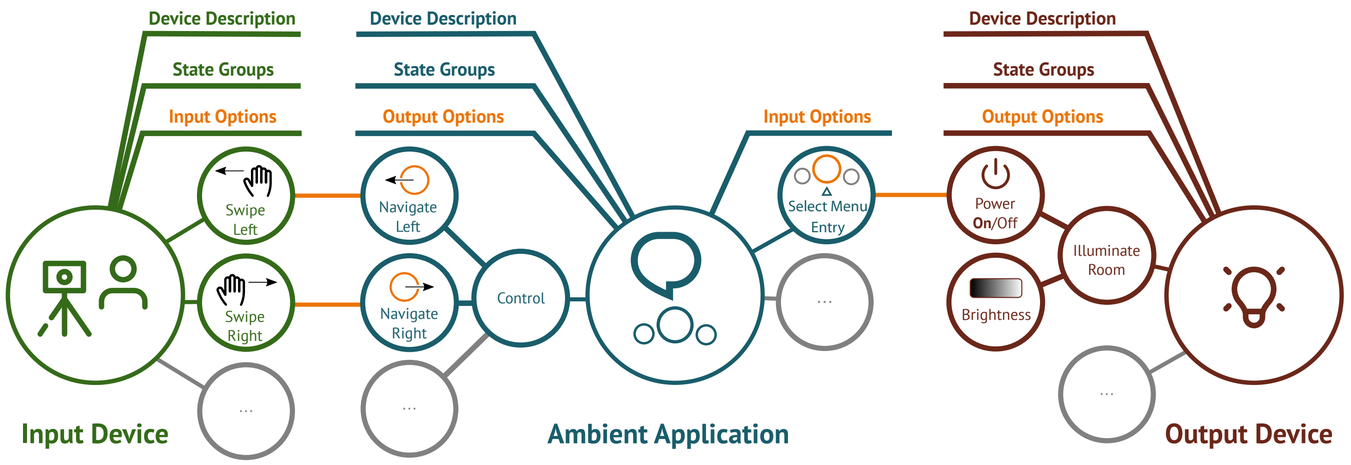


Figure 1: Example descriptions of an input device, an ambient application, and an output device, along with suitable connections: Descriptions consist of device or application details, state groups, as well as input and output options, grouped by tasks. Connections are formed between input and output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

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. Since 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*).

Since an ambient application is not necessarily directly connected to the physical world (e.g., when used as a tool for augmentative and alternative communication), 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 to omit the description of a real-world goal and to only describe tasks in these cases (like 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, like 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 A FRAMEWORK FOR SELF-REFLECTING APPLICATIONS

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 is able to provide such self-descriptions for dynamically coupled device ensembles based on the SODL. Besides, the framework addresses other self-* characteristics. On one hand, the system is self-organizing by being able to automatically create, restore and split device connections. On the other hand, it is also self-stabilizing since it is able to restore and also replace (parts of) device ensembles to keep the users able to solve their tasks. We extended it 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.

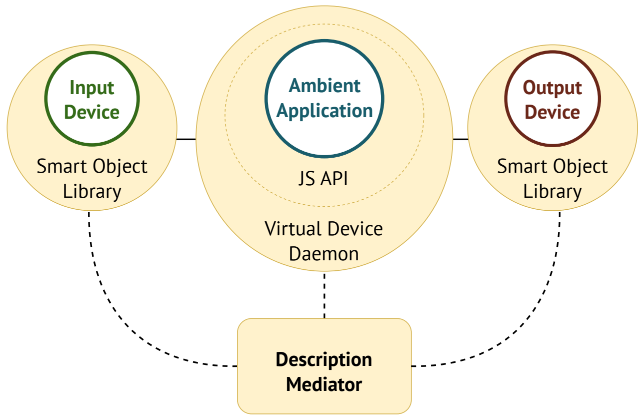


Figure 2: Example connections as realized by several components of the framework: Input and output devices use the Smart Object Library in order to forward their input and output to connected devices and ambient applications. Ambient applications running in the browser employ the Virtual Device Daemon via the Javascript API to do the same. Both devices and applications also provide a self-description to the Description Mediator, that establishes the connections in the first place.

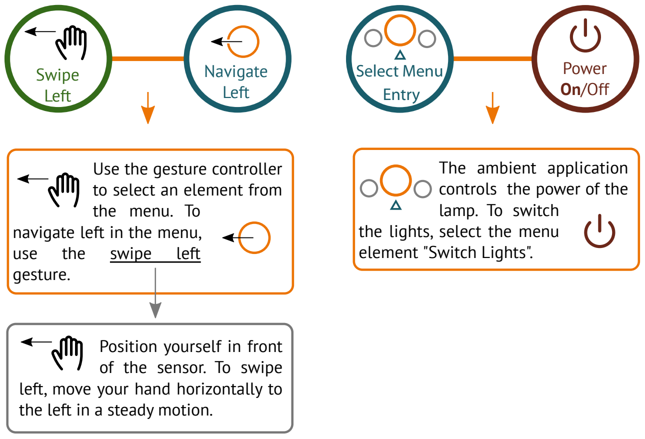
Basically, the framework consists of two components: Devices that make use of the device library, an easy way to integrate devices into the framework, and the description mediator which is used to connect devices based on their functionality, to generate merged descriptions, as well as to inform the 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 2).

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.

5.1 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 done automatically considering different extensible and configurable rules [6]. Towards this end, a probabilistic mediation method is usually used due to computational complexity. Alternatively, however, a mediation can also be specified by a direct configuration, which can be recorded, for example, in the input or output devices. An overall description of the system can ultimately be created based on the respective device descriptions. This is done 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.



Swipe Left
Navigate Left
Select Menu Entry
Power On/Off
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 3: 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

To realize all these features, the so-called description mediator, a component for the mediation, is used. We integrated the aforementioned changes to the SODL in the description mediator and its mediation process, so it is able to also connect ambient applications.

Figure 3 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).

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.

5.2 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. At start, 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 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 makes additional messages for information unrelated to the interaction relevant. 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.

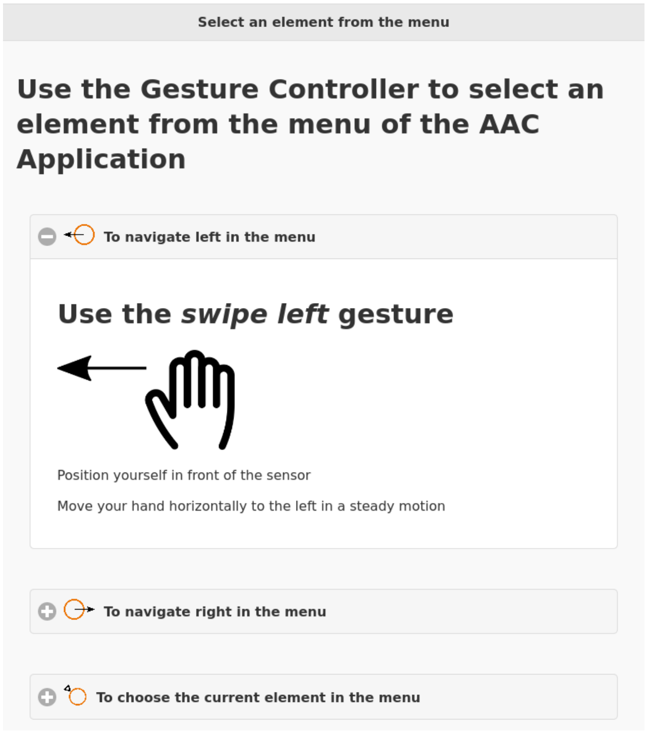
5.3 Virtual Device Daemon and API

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, since applications are usually implemented using other programming languages. In recent years, JavaScript-based applications were 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 like any other smart object by the framework. 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. Since the framework is based on standardized messages and those messages are forwarded also based on a standard, the API is easily extensible.

6 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,


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

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 device 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 4.

Interactive tutorials can be generated, as already previously described [7], 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 addressed better through tailored tutorials.

A general description of likely complex application logic using a formal language is in many cases neither suitable nor useful since a description on 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.

7 CONCLUSION

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 the 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, like 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.

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 usage of an application and device ensemble, similar to the tutorials described by Burmeister and Schrader [7]. We further plan to integrate the graphical representations of interaction techniques and the corresponding system response to provide more elaborated 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, like 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 and manage the set of apps currently active. 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 like other apps, 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 those apps, we currently investigate their explainability, since they can be used to provide rather complex interactions.

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

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.

ACKNOWLEDGMENTS

We thank numerous freeicons (https://freeicons.io) members for making their icons available through creative commons rights (CC BY 3.0, https://creativecommons.org): 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.

REFERENCES

[1] Bashar Altakrouri. 2014. *Ambient Assisted Living with Dynamic Interaction Ensembles.* Ph.D. Dissertation. Universität zu Lübeck.

[2] Serge Autexier and Rolf Drechsler. 2018. Towards Self-Explaining Intelligent Environments. In *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO).* 1–6.

[3] Daniel Burmeister. 2018. *Selbstreflexive Geräteverbünde in Smarten Umgebungen.* Ph.D. Dissertation. Universität zu Lübeck.

[4] Daniel Burmeister, Bashar Altakrouri, and Andreas Schrader. June 23 – 26 2015. 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).* ACM, Duisburg, Germany, 16–20.

[5] D. Burmeister, F. Burmann, and A. Schrader. 2017. The Smart Object Description Language: Modeling Interaction Capabilities for Self-Reflection. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops).* 503–508. https://doi.org/10.1109/PERCOMW.2017.7917614

[6] Daniel Burmeister, Bennet Gerlach, and Andreas Schrader. 2018. Formal Definition of the Smart Object Matching Problem. *Procedia computer science* 130 (2018), 302–309.

[7] Daniel Burmeister and Andreas Schrader. 2018. Runtime Generation and Delivery of Guidance for Smart Object Ensembles. In *Advances in Neuroergonomics and Cognitive Engineering (Advances in Intelligent Systems and Computing)*, Hasan Ayaz and Lukasz Mazur (Eds.). Springer International Publishing, Cham, 287–296. https://doi.org/10.1007/978-3-319-94866-9_29

[8] Anind K Dey. 2009. Explanations in Context-Aware Systems.. In *ExaCt.* 84–93.

[9] R. Drechsler, C. Lüth, G. Fey, and T. Güneysu. 2018. Towards Self-Explaining Digital Systems: A Design Methodology for the Next Generation. In *2018 IEEE 3rd International Verification and Security Workshop (IVSW).* 1–6. https://doi.org/10.1109/IVSW.2018.8494900

[10] Görschwin Fey and Rolf Drechsler. 2020. Self-Explaining Digital Systems: Technical View, Implementation Aspects, and Completeness. In *Advanced Boolean Techniques: Selected Papers from the 13th International Workshop on Boolean Problems*, Rolf Drechsler and Mathias Soeken (Eds.). Springer International Publishing, Cham, 1–20.

[11] Börge Kordts, Jan Patrick Kopetz, Katrin Balzer, and Nicole Jochems. 2018. 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.

[12] M. Kranz, P. Holleis, and A. Schmidt. 2010. Embedded Interaction: Interacting with the Internet of Things. *IEEE Internet Computing* 14, 2 (March 2010), 46–53. https://doi.org/10.1109/MIC.2009.141

[13] T. Kulesza, S. Stumpf, M. Burnett, S. Yang, I. Kwan, and W. Wong. 2013. Too Much, Too Little, or Just Right? Ways Explanations Impact End Users' Mental Models. In *2013 IEEE Symposium on Visual Languages and Human Centric Computing*. 3–10. https://doi.org/10.1109/VLHCC.2013.6645235

[14] Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. 2009. Why and Why Not: Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 2119–2128. https://doi.org/10.1145/1518701.1519023

[15] Jakob Nielsen. 1986. A Virtual Protocol Model for Computer-Human Interaction. *International Journal of Man-Machine Studies* 24, 3 (March 1986), 301–312. https://doi.org/10.1016/S0020-7373(86)80028-1

[16] Gerrit Niezen. 2012. *Ontologies for Interaction : Enabling Serendipitous Interoperability in Smart Environments*. Doctoral Thesis. Technische Universiteit Eindhoven, Eindhoven.

[17] Donald A. Norman. 2010. Natural User Interfaces Are Not Natural. *interactions* 17, 3 (May 2010), 6–10. https://doi.org/10.1145/1744161.1744163

[18] Neville A. Stanton. 2006. Hierarchical Task Analysis: Developments, Applications, and Extensions. *Applied Ergonomics* 37, 1 (Jan. 2006), 55–79. https://doi.org/10.1016/j.apergo.2005.06.003