User Interfaces for Smart Things

A Generative Approach with Semantic Interaction Descriptions

Simon Mayer*[1], Andreas Tschofen[1], Anind K. Dey[2], and Friedemann Mattern[1]

[1]Institute for Pervasive Computing, ETH Zurich
[2]HCI Institute, Carnegie Mellon University

April 4, 2014

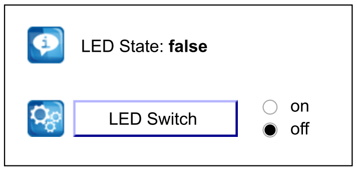
Abstract

With ever more everyday objects becoming "smart" due to embedded processors and communication capabilities, the provisioning of intuitive user interfaces to control smart things is quickly gaining importance. We present a model-based interface description scheme that enables automatic, modality-independent user interface generation. User interface description languages based on our approach carry enough information to suggest intuitive interfaces while still being easily producible for developers. This is enabled by describing the atomic interactive components of a device and capturing the semantics of interactions with the device. We propose a taxonomy of abstract sensing and actuation primitives and present a smartphone application that can act as a ubiquitous device controller. An evaluation of the mobile application in a laboratory setup, home environments, and an educational setting as well as the results of a user study highlight the accessibility of the proposed scheme for application developers and its suitability for controlling smart devices.

1 Introduction

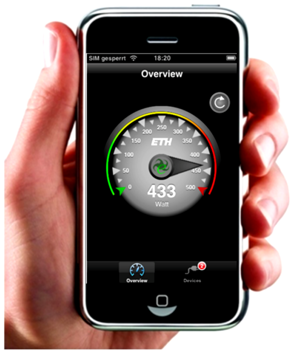
The *Internet of Things* advocates networked "smart things" that are enabled to interact and communicate with each other, with users, and with their surroundings and thereby become active participants in an environment of cooperative services and (physical) applications [13]. The success and widespread adoption

*This work is supported by the Swiss National Science Foundation under grant number 134631. Corresponding author: Simon Mayer, simon.mayer@inf.ethz.ch. Authors' addresses: S. Mayer, A. Tschofen, and F. Mattern: Institute for Pervasive Computing, ETH Zurich, Universitätstrasse 6, 8092 Zurich, Switzerland; A.K. Dey: HCI Institute, Carnegie Mellon University, NSH 2502C 5000 Forbes Ave., Pittsburgh, PA, USA.


LED State: **false**

LED Switch ○ on [✓ ● off]

(a)



(b)

Figure 1: a) Browser-based interface to toggle a Web-enabled LED [15]. b) Manually tailored mobile user interface of a smart electricity meter [30].

of the World Wide Web in linking documents and, later, in connecting individuals, has in turn driven a development commonly referred to as the *Web of Things*, which aims at re-using Web patterns and Web protocols to make networked physical objects first-class citizens of the World Wide Web [7]. In the Web of Things, smart things are viewed as hierarchies of resources that are uniquely addressable using Uniform Resource Identifiers (URIs) and whose interfaces comply with the principles of Representational State Transfer (REST) [5]. Leveraging the Web as an application layer for smart things allows for familiar mechanisms that made the Web successful to be extended to such real-world objects. Smart devices and their functionality can be directly referenced and bookmarked and can easily be incorporated in physical mashups using well-known and easy to use scripting languages or graphical programming concepts.

In the Web of Things domain, the main medium for a user to directly access and control a smart thing is the Web browser, a widely available tool that most users are familiar with [2, 4]. All user interaction happens via the thing's Web representation that displays sensed values and provides a simplistic, form-based, interface to control actuation of the smart thing (Figure 1(a)). While such an interface is easy to deploy or can even be generated automatically, it is often neither intuitive nor efficiently usable and the interaction itself may be cumbersome, especially when using a mobile device to interact with the smart thing. As a remedy, interfaces may be provided that are manually tailored to specific smart things (Figure 1(b)). However, these usually are expensive to create and not flexible enough to adapt to different platforms and scenarios.

To bring Web-enabled smart things into peoples' homes and enable humans to better interact with smart thing environments, more intuitive but still easily deployable interaction mechanisms are required. Here, our focus is on supporting *explicit* human interaction with smart things and thus emphasize the direct and immediate monitoring and control of such devices – a concept different from smart environments providing *invisible* background assistance. Such immediate interaction is relevant especially if the controlled devices provide information or perform actions of immediate value to the user (e.g., monitoring electricity meters or controlling multimedia systems).

To enable the *automatic generation* of user interfaces for smart devices, a model-based approach seems to be best suited [21, 6]: the smart thing embeds a description of how other devices can interact with it in the form of a User Interface Description Language (UIDL), and interaction devices (e.g., remote controls or smartphones) use this information to provide an appropriate concrete interface to the user. Such a system enables plug-and-play interaction within smart things environments with no configuration effort other than embedding the appropriate descriptions. Thereby, it greatly reduces the amount of time and work needed to create appropriate and easily usable user interfaces for smart devices [21].

In this paper, we present a high-level description scheme for smart things that captures the semantics of an interaction with the device rather than providing an explicit, concrete encoding of an appropriate type of user interface or its appearance. Based on this approach, we propose a modality-independent taxonomy of interaction semantics for monitoring and controlling devices. Furthermore, we present a concrete description language that captures interaction semantics and, based on this language, a prototype implementation of a universal remote control application for smartphones. This application as well as our description scheme have been evaluated within a controlled laboratory environment, with mock-up smart devices, in deployments in private homes, and in a study that targeted the understandability and simplicity of the proposed description scheme.

Our focus is to allow the provisioning of interaction descriptions for smart devices by adding a minimal amount of markup that is simple to produce and easy to understand for developers. Still, the description scheme is general enough to be applicable to a wide range of interaction use cases, where we primarily consider devices that monitor and control physical quantities in the real world. Such devices are widespread in home and building automation systems (e.g., light dimmers, window blind motors, or household appliances) but can also be found in cars (e.g., air conditioning), electric musical instruments, toys, and many other devices that we interact with in our daily lives. More and more, these traditionally simple, isolated devices are being equipped with processing and communication capabilities, thus transforming them into smart things. Since the main functionality offered by such devices is to sense and/or actuate the physical world, they can be modeled as actuators, sensors, or sensor-actuator-composites. A light dimmer, for example, is a simple actuator that controls the electric power supply of a lamp and consequently its brightness. A toy robot, in contrast, might have multiple motors to control its movable parts, and sensors to perceive its environment.

After discussing the terminology used throughout this document, we introduce our approach of describing the high-level semantics of interactions in Section 3. We detail how these interaction semantics can be captured in Section 4 and discuss elements of a language to describe them in Section 5. In Section

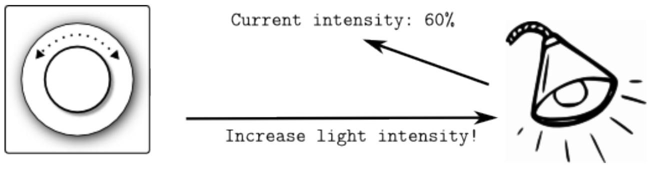


Figure 2: An interactor (light dimmer) and a stateful atomic interactive component (dimmable lamp) whose state can be queried and manipulated.

6 we show a prototype application that interprets our interaction descriptions and can be used as a generic mobile user interface in smart environments. We present an evaluation of our language with respect to its generality, usability for end users, and producibility for developers in Section 7 and discuss the positioning of our approach with respect to related work in Section 8. Finally, we provide our conclusions and highlight avenues for potential future work in Section 9.

2 Terminology

Throughout this paper, we consider an *atomic interactive component* to be a physical or virtual object that provides a specific functionality to the user and is not reasonably subdivisible (e.g., a light switch). In contrast, a more complex device like a VCR can be subdivided into multiple atomic interactive components, for instance its *Play*/*Pause* buttons and its volume controller. An atomic interactive component either provides its current internal state as data (sensor) or performs an action when reacting to a command (actuator).

We differentiate between two types of actuators: A *stateful actuator* is an actuator whose state can, in addition to being manipulated by sending commands, also be queried. One example for this is a dimmable lamp (cf. Figure 2) whose state is the lamp's current intensity/brightness. A *stateless actuator* can be manipulated like a stateful actuator but does not hold a representation of its internal state: tt can be triggered but not queried. An example is a digital doorbell that plays a sound when triggered by a button press. Every stateless actuator can in principle be transformed into a stateful actuator by exposing its current state. However, for this type of actuator, representing the state is often not necessary or results in too much overhead because the state does not play an important role or is hard to capture.

The device or software abstraction that a human uses to interact with an interactive component is called an *interactor*. In the example depicted in Figure 2, this is the light dimmer knob. Interactors can take many forms ranging from traditional graphical user interfaces (GUIs) to gesture or speech-control interfaces and to physical buttons or knobs.

3 Interaction Semantics and Atomic Interactive Components

We postulate that an interaction description scheme should be usable by heterogeneous interaction devices, involving gesture-based, speech-based, graphical, or physical interfaces like physical buttons or sliders (see [25]). Furthermore, to foster widespread adoption and actual usage in practice, a user interface modeling language should, in addition to being expressive enough, be easy to understand for developers (see [12]). Ideally, it should thus enable the embedding of interaction descriptions with only a few lines of easily producible markup.

Approaches to providing UIDLs proposed to date are limited in their support for these requirements. Most target the provisioning of interface descriptions for complete devices (see Section 8 for a thorough discussion of related work): the user interface of a VCR, for instance, is usually specified in a single document that, apart from describing the interfaces to each of the VCR's components (e.g., the *Play*-button), lists all dependencies between components of the device. An example of such a dependency is that the fast-forward button of the VCR should only be active when a video is playing. Instead of describing devices as a whole in this way, we claim that, especially within a Web context it is more beneficial to embed interaction information directly into the devices' atomic functional components and not explicitly specify such dependencies. Other projects in the domain of UIDLs have also advocated the decomposition of appliances into their atomic interactive components (e.g., XWeb [24], the URC standard [31], the PUC project [21], or MARIA [27]), but have not explored this further as an approach that could yield simpler yet expressive interface description languages.

The traditional way of describing how one can interact with atomic interactive components is to associate them with data type information: an element which has an *integer* or *float* type with a range can be graphically represented by a slider; an *enum* type corresponds to a dropdown menu, and so on. We argue that providing such a data model, however, is only a specification of the program interface to an interactive component. While this enables rudimentary interaction with devices, the specification of data types is not sufficient for creating intuitive interfaces which, in our opinion, requires capturing the *semantics* of the interaction. Therefore, we propose to abstract from *user interface descriptions* to *interaction descriptions*, meaning that we do not model concrete interface elements but instead the semantics of the interaction of the user with a device (what exactly we mean by *interaction semantics* is discussed in detail in Section 4.2). The possibility of adding more abstract information about interface elements has also been expressed by the authors of the aforementioned PUC papers and the URC standard, but has not been further explored as a possibility to simplify user interface descriptions.

When analyzing traditional (built-in or remote) user interfaces for devices like the ones mentioned above, one notices that certain types of interactors (e.g., various kinds of knobs or combinations of buttons) occur again and again but control very heterogeneous types of actuators and sensors. For instance, from a user interface point of view, a light dimmer knob and a thermostat knob are equivalent: although certainly a little unusual, it would be possible to control a lamp's brightness with a thermostat knob. This interchangeability is not confined to interactors with similar physical appearance. In fact, both brightness and temperature could also be controlled by a graphical slider widget, or with speech commands ("increase/decrease brightness/temperature"). The reason for this is that the semantics of interaction are the same for the two interactive components: in both cases, the user scales a physical intensity.

We argue that this observation can be generalized: in the following, we propose a classification of interactive components into semantic interaction categories that suggest appropriate interactors but are still general enough to be applicable for a wide variety of smart things. Interestingly, although this idea originates from the analysis of physical actuators and sensors, it can also be applied to a range of appliances and software applications whose actuators and sensors are virtual, because they employ physical metaphors (e.g., switches or scroll bars) for their control. We hence take a different point of view on user interfaces: rather than considering a "knob" as a physical entity that is emulated in GUIs, we consider the interaction semantics behind knobs and argue that such "conceptual knobs" occur in different forms which all encode the same interaction intention, namely scaling a value between two extremes. Furthermore, we show that different interaction semantics do not exist by themselves but rather can be arranged in a semantic interaction hierarchy, with the most specific interaction types that carry most semantic constraints at the bottom and the most generic interaction types at the top.

Building on these concepts, we suggest that it is possible to create interfaces that represent a substantial improvement over traditional type-based widgets by implementing only a relatively low number of abstractions to capture interaction semantics. This can be leveraged to reduce the developers' effort for providing interface descriptions and thus fosters their widespread adoption. Furthermore, specifying interaction descriptions on this more abstract level allows human interaction with smart things regardless of the type of interaction device or the modality of the interaction (e.g., gestures, haptics, or speech). Finally, the selected level of abstraction and the hierarchical structuring of interaction semantics enable the interaction device to adapt the manifested interface to its own capabilities and/or user preferences. For instance, an interactor without a graphical interface could still be able to generate an interface for switching between different modes, or for scaling (e.g., using its gyroscope). We thus claim that a scheme which describes interactions semantically is, in general, more suitable to support user interaction within smart things environments than other approaches to user interface modeling proposed in the literature (e.g., [21, 24, 27, 31]).

Restricting our scheme to the description of the atomic interactive components (in our case, URI-identified resources) of a smart thing leads to a low entry barrier for users (e.g., Web developers) to create interaction annotations for devices. Our approach is thus well-adapted to the devices that are our main concern: in the context of the Web of Things, smart things are rather simple devices whose capabilities (i.e., interactive components) are hierarchically structured due to the adoption of a hierarchical resource-oriented architecture (REST, see [7]). For this reason, it is possible to view such a device as a structured collection of its sensors and actuators rather than as a single entity that supports complicated tasks.

However, even if interactive components of a smart thing are described independently, they can still be aggregated within composite user interfaces without explicitly modeling semantic relationships between components. This is the case because the user interface should usually only reflect a devices' internal state and not introduce further constraints. As an example, consider a simple user interface for a television set that comprises a volume level and an on/off switch. Showing the volume level interface as inactive when the device is switched off is then not a constraint that is introduced by the user interface but rather reflects the fact that the volume cannot be set without switching on the television set. In our proposed concept, one would independently describe the switch and the volume level and leave the decision whether the volume level is active or not to the controlled device itself rather than to the interactor that renders the user interface. We thus argue that the proposed language can also be used to describe interfaces for composite devices with dependencies between components and refer to the discussion of this property in Section 7.

Apart from keeping descriptions simple to understand and produce, the decomposition of devices into their atomic interactive components has further advantages regarding the generation of user interfaces: while it remains possible to create thing-centric interfaces (i.e., to meld all capabilities of a device into a single interface), the proposed approach adds the possibility to create task-centric interfaces by integrating components of different smart things within a single user interface. Examples for interfaces that are tailored around a specific task are an interface that displays information from all temperature sensors within one building or an interface geared toward watching a movie that incorporates sensors and actuators from the television, stereo, and DVD player. Thus, this property allows different devices or persons to view "their" smart environment from different perspectives.

4 Capturing Interaction Semantics

In classical model-based approaches, interactive components are modeled mainly or exclusively using data types. While our proposed description methodology also makes use of data type information, it merely does so to describe the exchanged data and not the user interface of the component per se. In fact, we propose to describe components by using interaction descriptions that consist of two parts, *data type information* for the data exchanged with the component and information about the *high-level semantics* of the interaction.

4.1 Data Types

In the proposed interaction description format, every atomic interactive component has an associated data type that represents the type of the entity state for sensors and stateful actuators. For stateless actuators, it gives the type of the argument that should be supplied when triggering the actuator. Our description supports the data types *boolean*, *integer*, *number*, *enum*, and *string*, where the well-known types have the usual semantics. *enum* requires the definition of allowed values, which can be given as a static list of values or as a list referenced by a URL for dynamic sets of allowed values. Optional arguments include defining the value's *unit* and the *allowed values* or *range*. Allowed values for the *string* type can be described as a regular expression or a definition according to the *JSON Schema*[1] format. Again, we stress that these types are not given to derive an appropriate user interface but rather to populate derived interfaces with meaningful values.

4.2 Semantic Interaction Abstractions

Knowledge about the data type already specifies how to interpret the component's state and which values are allowed as state and thus enables basic type-safe interaction. Considering as an example a window blind controller with the states "down," "stop," and "up," a graphical interaction device could, for instance, generate a drop-down list using only the type information (in this case, *enum*). However, this interface is hard to understand and use, and unnatural: it demands that users perform the mapping of their interaction intent (moving the blind) to the choice of a state of the blind motor. Furthermore, to bring the blind down just a little bit, the user would have to open the drop-down list twice in rapid succession to start and stop blind movement, respectively. These problems arise because the data type is only a specification of the *program interface* to an interactive component and does not consider the semantics of the interaction. As a way of capturing these interaction semantics, we propose the concept of *semantic interaction categories* for interactive components, which we call *interaction abstractions*. We have identified about a dozen distinctive interaction abstractions that capture interaction semantics related to sensing as well as stateless and stateful actuation. Clearly, not every interaction with any smart thing falls into one of these categories and the classification should, therefore, be considered as a proof of concept that can be extended as required. However, we found that the proposed categories already cover all of the use cases that we consider within our deployments. The set of abstractions itself was obtained by considering typical devices in several core domains where smart things play, or are supposed to play, an important role:

- Home and building automation systems: lighting, HVAC, curtain & blind control, audiovisual equipment, security, electricity metering, and so on.

[1] `http://json-schema.org/`

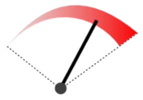
Name	Example Symbol	State Description	Example
`get data`		General data.	Display current song.
`get value`		Ordered domain.	Display temperature.
`get proportion`		Ordered domain with fixed range.	Display load of a server.

Table 1: Interaction abstractions for sensors

- Home and office appliances: washing machines, coffee machines, and so on.
- Auditorium control systems: lighting, A/V selection and controls, controls for peripherals like blackboards or projectors.
- Cars: air conditioning, drive controls, comfort controls, and so on.
- Public services: ticket machines, vending machines, and so on.
- Electronic toys and musical instruments.

Three of the proposed interaction abstractions apply to sensors, two to stateless actuators, and eight to stateful actuators. For sensors (Table 1), an interaction abstraction captures the nature of the measured data. The `get proportion` abstraction, for instance, suggests that the value measured by a sensor should be considered with respect to its possible range and rendered appropriately, for instance as a progress bar or gauge. For actuators (Tables 2 and 3), the interaction abstraction stands for a primitive of actuation that provides information with respect to three dimensions: the *abstraction of actuation*, the *semantics of values* in the domain of the actuator, and the suggested *interaction pattern*. As an example, consider the `move` abstraction (cf. Table 3) with data type *enum* for the window blind controller that was described before:

Name	Example Symbol	Description	Example
`trigger`		Trigger an action.	Reset button.
`goto`		Adjust one-dimensional state.	Change track on hi-fi unit.

Table 2: Interaction abstractions for stateless actuators.

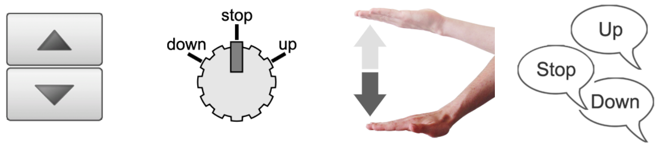


Figure 3: Example user interfaces for (vertical) `move` actuators.

- The *abstraction of actuation* refers to the physical (or metaphorically physical) actuation that the actuator can perform. The `move` interaction primitive, for instance, implies that the performed actuation refers to a movement.
- The *semantics of values* associate the values in the domain of the interactive component with a meaning or function. For `move`, the domain is ordered with a neutral value in the middle, the neutral value corresponds to no movement, and values below and above correspond to a movement in one or the other direction.
- Concerning the suggested *interaction pattern*, a `move` interaction is usually composed of two parts: starting and stopping the movement. An interactor can implement this by, for instance, falling back to the neutral value (*stop*) when the user activity ends (e.g., when the user releases the corresponding button).

Some interactors that satisfy these requirements for the `move` abstraction are shown in Figure 3. Other components with the interaction abstraction `move` are, for example, a robot arm motor, or an actuator that supports rewinding and fast-forwarding a video (by "moving" the current point in time). As Figure 3 suggests, interaction abstractions are modality-independent and can be mapped to graphical widgets, physical interactors, and speech or gesture commands.

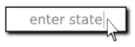
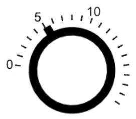
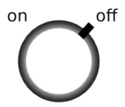
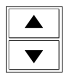
Name	**Example Symbol**	**State Description**	**Example**
`set`		General data.	Set text displayed on a screen.
`set value`		Ordered domain.	Set minutes until alarm.
`level`		Ordered domain with a neutral value.	Scale an image.
`set intensity`		Ordered domain with fixed range.	Set loudspeaker volume.
`switch mode`		Operating mode.	Switch ventilation mode.
`switch`		On or off.	Switch on/off lamp.
`position`		Point in one-dimensional space.	Position window blind in one-dimensional space.
`move`		One-dimensional movement.	Move window blind.

Table 3: Interaction abstractions for stateful actuators.

Furthermore, a particular interaction abstraction usually can be superimposed on interactive components of multiple different data types. This is the case because it encodes the high-level semantics of an interaction, while the data type depends on the implementation of an interactive component. As an example, a more sophisticated blind control could be of type *integer* (instead of *enum*), where the absolute value of the state would correspond to the speed of the blind movement. Still, the appropriate interaction abstraction for this controller is `move`.

The interaction abstractions can be organized in hierarchical taxonomies (Figure 4), where the root abstractions `get data`, `trigger`, and `set` are the most abstract ones possible. These three abstractions yield simple, solely type-based interactors. Descending within the hierarchy, the semantic information

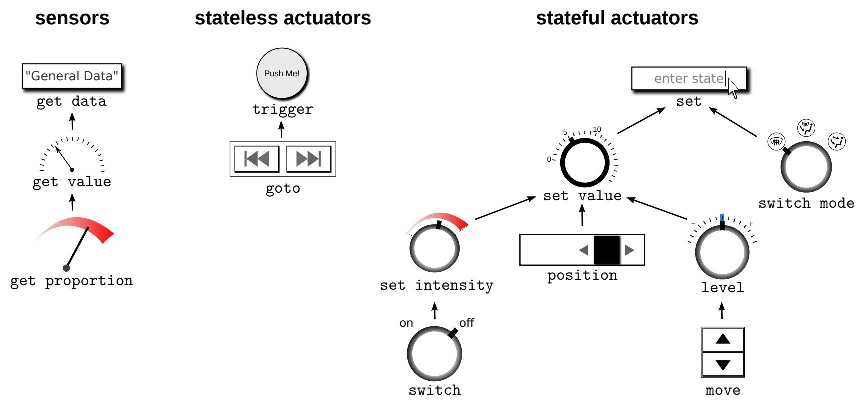


Figure 4: Interaction taxonomies for sensors, stateless actuators, and stateful actuators.

gets more concrete. As an example, consider `level`, the parent abstraction of `move`. It suggests that the domain of the actuator's state is ordered with a neutral value which should be reflected by corresponding interfaces (e.g., a knob that snaps into place in the middle position). The `move` abstraction adds information about the effects of setting this state (a movement).

The hierarchical organization of interaction abstractions can be exploited as a fallback mechanism: if an interaction device does not know or does not support a particular interaction abstraction, it can traverse the tree upward until it finds a more general abstraction that it can handle. This enables the scheme to also be used by simple interactors that only support a subset of the available abstractions. Furthermore, the taxonomy stays extensible as new abstractions do not have to be known by all interaction devices from the beginning. One can thus build large taxonomies with arbitrarily specialized interaction abstractions at their leafs.

Within our implementation of this concept, we aimed at generating small taxonomies to make the descriptions easy to understand, embed, and interpret for developers. We found, however, that the small set of abstractions proposed earlier already covers a large set of interactive components (cf. Section 7).

5 Elements of a Semantic Interface Description Language

The classification of interaction semantics together with the described taxonomy can be used to define a concrete semantic interface description language by embedding information about the appropriate data type and interaction abstraction as metadata into the interactive components of devices. In this section, we present elements of an implementation of such a language that is based on the interaction abstractions shown earlier (Tables 1-3), and that allows user interface devices to render interactors when confronted with a corresponding interactive component. This language was used in a prototype implementation of a generic mobile user interface for smart things, which is described in Section 6.

For reasons of legibility and understandability, the reference implementation of our proposed language is shown in JavaScript Object Notation[2] (JSON). A description of the window blind controller mentioned above, for instance, then looks as follows:

```
{
  "type" : {
      "name" : "enum",
      "values" : ["down","stop","up"]
  },
  "abstraction" : {
      "name" : "move",
      "orientation": "vertical"
  }
}
```

The proposed concepts of our language can, however, be expressed and embedded in multiple formats, for instance as XML documents or as HTML-based Microformats markup. For annotating Web-based smart things, we found HTML Microdata[3] to be particularly convenient because it allows to embed information that is meaningful for humans and machines within the same document. Using Microdata thus allows us to create a document that looks like an ordinary Web site to humans but still contains all information that is necessary for interaction devices to generate user interfaces. The following markup illustrates how to embed the aforementioned description as Microdata in the HTML representation of a device:

```
To <span itemprop="name">move</span> the blinds <span
    itemprop="orientation">vertical</span>ly, set the
    controller to one of the values <span itemprop="type-
    range">[down,stop,up]</span>  (data type: <span
    itemprop="type-name">enum</span>).
```

An alternative to this direct embedding of interaction metadata in the Web representations of our smart things is to store the descriptions on a remote server and have devices provide links that point to them, for instance using Web Linking[4].

Our proposed interaction description language has been built around a small kernel that consists of data type and interaction semantics information to keep descriptions simple to understand and write for humans, and easy to parse for machines. To enhance the experience of using an interface created from these

[2]http://www.json.org/
[3]http://w3.org/TR/microdata/
[4]http://tools.ietf.org/html/rfc5988

descriptors, we additionally propose optional properties that allow for the refinement of the interaction abstraction, for instance, to specify details of the actuation or to define dedicated values. We briefly describe two such properties that we found to be particularly helpful for augmenting automatically generated interfaces, *anchors* and *orientation*. *Anchors* can be used to add special meaning to certain values or value ranges. For instance, specific values (e.g., the range of 95% to 100% when displaying the load of a server) can be marked as potentially harmful (or particularly desirable), which then can be reflected in the user interface. Similarly, especially for the `move` and `position` abstractions, the desired interface orientation can be specified, for instance as *vertical* for a window blind controller. Additional properties could be defined for increased customization of user interfaces – doing so excessively, however, could potentially corrupt the language's simplicity. To allow for the possibility of manually tailored user interfaces and the mixing of these and automatically generated interfaces, our proposed description scheme also includes the possibility of specifying links to dedicated Web interfaces that can be displayed by interaction devices.

We used JSON Schema, a specification used to define the structure of JSON data, to create a formal definition of our language. This definition includes a human-readable documentation of the language and contains all information necessary for structural validation of interaction descriptions, which is useful in automated testing. To give an example of such a specification, the following JSON Schema document shows the structural definition of the `move` abstraction:

```
{
  "name":"move",
  "description":"Temporarily level a value of a device
      property negatively or positively. Implies a
      virtual or physical movement.",
  "type":"object",
  "properties":{
    "unit":{
      "type":"string"
    },
    "neutral":{
      "type":["string","number"]
    },
    "default increasing value":{"$ref":"default inc.
        value"},
    "default decreasing value":{"$ref":"default dec.
        value"},
    "anchors": {"$ref":"anchors"},
    "orientation": {"$ref":"orientation"}
  }
}
```

The schemas that are required to validate an interaction description using this document (e.g., *anchors* or *orientation*) are defined in separate definitions that are not reproduced here.

6 A Generic Mobile User Interface for Smart Things

To evaluate the discussed concepts in practice, we implemented a prototype application for mobile devices running the Android operating system. This application interprets our interaction descriptions (see Sections 4 and 5) and allows end users to interact with devices via automatically generated interfaces. End users can also store interfaces locally on their interaction device and can aggregate multiple of these within composite interfaces (i.e., as widget lists). Specifically, using the application, users can create new task-centric composite interfaces and give them a name that one can better relate to, such as "My Lecture Hall Controls." To populate a composite interface with widgets, users can then select from the stored interactors. Composite interfaces can also be associated to specific locations; for instance, an individually tailored lecture room interface could be loaded whenever the user enters that room.

Our interactive components embed interaction descriptions as Microdata within their HTML representations, which is mapped to our JSON reference format using a discovery service that can handle smart things with embedded semantic descriptions, presented in [14]. When the mobile application discovers such a component, it retrieves its interaction description and instantiates an appropriate interactor. If multiple suitable interactors are found, one of them is rendered and the user is given the opportunity to browse through the other interactors. From this point on, the application uses HTTP requests to get and update the state of the component by exchanging plain values that correspond to the type definition of the interactive component. Our application immediately provides multiple different interactors to users because the prototype was created to load interfaces with different interaction modalities for demonstration purposes. For a final system that is used by end users, a single interactor could be rendered, with the possibility of loading more interfaces if desired by the user.

We implemented various graphical interactors (e.g., gauges, click wheels, knobs) that correspond to the defined abstractions (see Figure 5 for some examples). Some of these also capture the optional definitions (for instance, a knob with a value range as *anchor* that causes vibration and turns red when this range is entered). Furthermore, we used smartphones as haptic input devices and mapped interaction abstractions to physical movements of the handset or interaction with the touchscreen. For instance, the handset can be tilted or turned like a knob to switch between operation modes or to move a robot arm (see Figure 6). One can `trigger` or `switch` by shaking the handset, and by swiping over the screen, one can use the `goto` abstraction for stateless actuators. The implemented generic mobile user interface for smart things shows that it is indeed possible to map the interaction abstractions to heterogeneous, also nongraphical, modes of interaction.

To further explore the modality-independence of the proposed description scheme, we also investigated speech-based interaction where appropriate speech

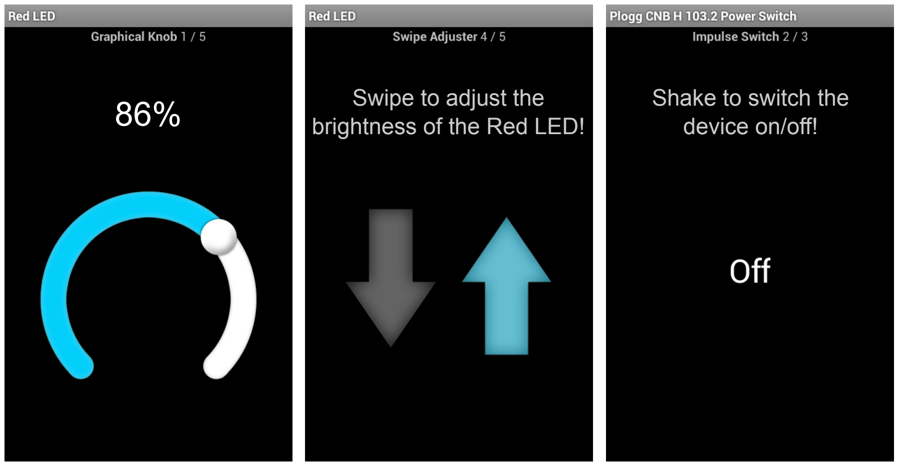

Red LED
Graphical Knob 1 / 5
86%
Red LED
Swipe Adjuster 4 / 5
Swipe to adjust the brightness of the Red LED!
Plogg CNB H 103.2 Power Switch
Impulse Switch 2 / 3
Shake to switch the device on/off!
Off

Figure 5: Smartphone interfaces for controlling the brightness of a LED (“`set intensity`”, left and middle) and a power switch (“`switch`”, right).

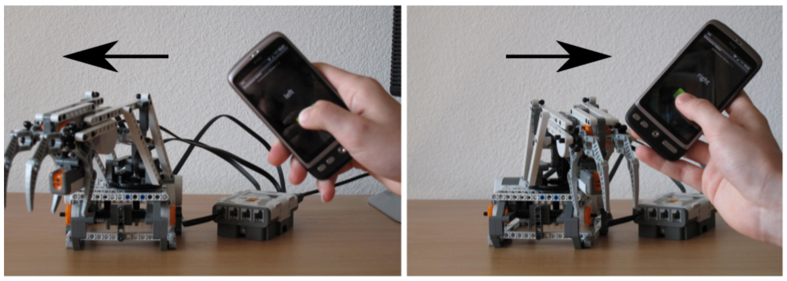


Figure 6: Rotating the smartphone to move a toy robot arm (“`move`”).

commands are inferred based solely on the semantics captured by the interaction abstraction. The only additional information needed to enable actuation commands is related to the selection of the interaction target (e.g., by using the target’s name): for a volume controller, the target could be “Volume”, for a window blind motor, “Blind”; and so on. This information is not captured by the interaction abstractions, as it is specific to every single actuator and determined by user preferences. It can thus be provided either by the interactive component (e.g., as an additional *name*-annotation) or alternatively by giving the user appropriate means to name smart devices himself. We defined speech commands such as “move *target* up/down” and “stop *target*” for vertical move actuators. Note that, besides leveraging the semantic information of the interaction abstractions to choose appropriate command phrases to listen for, we can also benefit from associated information about the interaction pattern. For instance, the pattern in a `move` interaction is to start the movement and then stop it again when the moved object has reached the desired position. For speech interaction, this pattern translates to listening for a "move *target* up/down" command, followed by a "stop *target*" command rather than only a single command, such as, for instance, for a `switch`. Finally, the automatically created speech interfaces also allow users to select their desired mode of interaction, for instance by choosing which of the commands "increase/decrease *target*" or "set *target* to *value*" to use for `set value` interactors. We implemented speech-based interfaces for all proposed interaction abstractions in the smartphone application and found that, in all cases, these were appropriate and easy to use for controlling smart devices.

To start controlling a smart device, the smartphone must initiate a connection with the device and retrieve the embedded information about its interface. Because our prototype application deals with smart things in a Web context, we assume interactive components to have a Uniform Resource Locator (URL). Given such URLs for smart things, resource association in our mobile prototype application can be performed by various means: apart from manually entering the URL of the object to interact with, the application is able to decode 2D barcodes that encode device URLs. Furthermore, when installed on a device that features an active Near Field Communication (NFC) component, an appropriate interactor is displayed automatically when the phone comes close to an NFC tag that encodes a resource URL. Finally, by making use of geographical location information offered by many of our prototype devices, we have added context-sensitive behavior to the application: using the GPS module of the mobile device, interfaces that are stored on it are ranked with respect to their distance to the user and the closest interfaces are directly presented in the application's main interface. Operation within secured environments is enabled by prompting the user to enter a username and a password to interact with restricted device components.

7 Evaluation

The proposed description language and the mobile prototype application have been deployed in various environments to determine whether our approach is general enough to capture the interaction semantics of typical devices in Internet of Things scenarios. We demonstrate the generality of the language by discussing the multitude of devices that were annotated using the language and give details about the concrete deployments in our laboratory and in private homes in Section 7.1.

Next, in Section 7.2, we demonstrate that our proposed interaction descriptions are producible not only by experts who are already familiar with our system but also by individuals with little prior training. To show this, we conducted a user study among 780 students from all faculties of our institution whose task was to create interaction descriptions for 19 scenarios from the home automation domain. This study thus elicits the developers' perspective with respect to the creation of user interface descriptions.

Finally, in Section 7.3, we show that the interfaces that are generated from our descriptions can be efficiently used by end users. After discussing how users interacted with annotated devices from our deployments, we elaborate on two case studies: the first shows how our language can be used in the context of a lecture hall control system. Here, we particularly consider the creation of composite, task-centric, user interfaces. The second case study, a user interface for a music player, demonstrates that our interaction descriptions can be used to create user interfaces for devices with a complex internal state, even though they do not explicitly model dependencies between interactive components of a device.

7.1 Assessing Generality: Laboratory and Real-World Deployment

To assess the generality of our proposed language, we first deployed the system in a laboratory environment. We added interaction annotations to several existing deployments of smart things present in our lab: SunSPOT sensor nodes (sensors: temperature, light, acceleration, orientation; actuators: tri-colored LEDs), smart plugs (sensor: electricity meter; actuator: power switch), a toy robot (sensor: ambient light; actuators: three motors), a remote-control toy car, mobile loudspeakers, and a smart thermostat. In addition, we created mockup implementations of a home automation system with embedded interaction annotations (lighting, blinds, stereo set, TV) and a lecture hall control system (volume and microphone controls, lighting control, A/V, peripherals, etc.). Specifically for the lecture hall controls, we modeled the exact capabilities of the system installed in our institution's lecture halls, which had not been considered earlier when designing the interaction abstractions. To evaluate the performance of the scheme with respect to more complex devices, we also created a Web proxy for the iTunes music player application and modeled interfaces to its functionality (e.g., controlling playback, adjusting the volume, choosing a track to play from a playlist) using our interactions markup. Both the lecture hall controls and the music player are discussed in more detail in Section 7.3.

Our scheme and the prototype application were also tested outside of a laboratory setting, as two members of our research group deployed the system in their private homes. These individuals have used the system about once per day, to control entertainment equipment and smart electricity outlets with metering and switching capabilities. At the time of writing, one of the private deployments still exists and has been running for a year (intermittently). We found that our proposed interaction descriptions covered all sensors and actuators present in our laboratory deployment, in the home automation scenarios, the lecture hall control mockup, and the music player.

7.2 Assessing Producibility: User Study

Apart from exploring whether our language is suitable for describing user interfaces in our use case scenarios, we also investigated the producibility of our proposed interaction description language by creators of user interfaces for smart things. A preliminary evaluation showed that for members of our research group it was easy to create the interaction markup for typical use cases: after a 2-minute introduction to the language, these individuals were able to apply the data type information and interaction abstractions to all devices and software components described earlier. However, to assess the accessibility of the description scheme and the ease of creating such interaction abstractions for individuals that had not been exposed to the system before, we conducted an online user study. Participants ($N = 780$) were asked to select appropriate interaction annotations and data types from the set described in Section 4 for 19 different scenarios ranging from a simple doorbell button and lighting scene controls to the description of interactive components of a VCR. Our test concentrates on the selection of appropriate abstractions and data types instead of having the participants implement a description because the implementation step can be fully automated by providing an application that takes the selected abstraction/data type pairs as input and produces annotations in the JSON reference format (see Section 5).

The study participants were students from all faculties of ETH Zurich with a self-reported average proficiency with Information and Communication Technologies (ICT) of 3.65 ($SD = 0.82$) on a 5-point Likert scale (1=No Knowledge, 2=Basic Knowledge, 3=Good Knowledge, 4=Advanced Knowledge, 5=Expert Knowledge). The participants had no prior knowledge of our project and no training with using the interactions scheme. They were presented with a 1-page description and reference document during the survey, which they were asked to study for about 2 minutes before working on the scenarios. For every scenario, the participants were asked to complete four tasks: (1) Select the most appropriate interaction abstraction, (2) select the appropriate data type, (3) give a confidence level for their choice in (1), and (4) give a confidence level for their choice in (2), where they selected the confidence levels on a 5-point Likert scale (1=Not confident at all ... 5=Very confident). We also recorded the time taken by the participants to work on the individual scenarios.

Across all scenarios, the participants selected a correct interaction abstraction in 84.2% ($SD = 6.99\%$) of the cases, on average. To do all four tasks associated with a scenario, they needed 40.3s, on average, where the high standard deviation of 16.7s is to a large part due to the high average time of 93s that participants spent working on the first scenario (a doorbell button) and can thus be largely attributed to habituation effects, as the order of the scenarios was not randomized. The self-reported confidence level of the participants was 4.09 on average, with minor fluctuation across all scenarios ($SD = 0.26$). Evaluating the performance of participants with an ICT proficiency of (5/Expert Knowledge) ($N = 170$) separately revealed that these performed a little better than the average, selecting an appropriate abstraction in 88.7% ($SD = 10.47\%$)

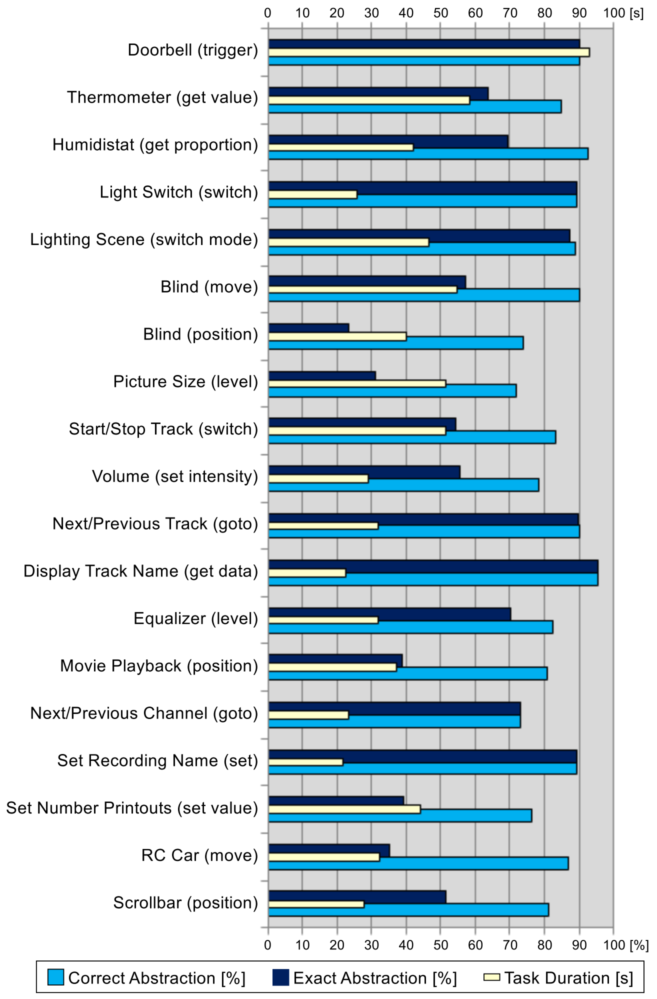


Figure 7: Performance and timing values for each of the 19 scenarios.

of the cases (16.9 correct answers out of 19).

Figure 7 shows the average performance of the participants as well as their timing values for each of the 19 scenarios. We present two distinct values to assess the participants' performance: the values for *Exact Abstraction* show how many of the study participants selected the interaction abstraction which captures best (in our opinion) the semantics of the described scenario, while the *Correct Abstraction* designates the percentage of participants who selected a suboptimal abstraction that is appropriate for the scenario but captures less of the semantic information. For each scenario, the *Exact Abstraction* is stated in brackets. In general, the set of *Correct Abstractions* for a given scenario is the *Exact Abstraction* plus all abstractions on the path of the most appropriate abstraction to the tree root in the taxonomy (see Figure 4). From the data, one can see that, for some of the scenarios, people strongly agree with each other and with our assessment concerning the type of abstraction to be used (e.g., for the *Light Switch*, *Lighting Scene*, and *Display Track Name* tasks). For others, though, there is considerable disagreement about which abstraction of interaction to use. As an example, consider the *Picture Size* scenario, where participants were asked to specify the appropriate abstraction to set the zoom level for a digital picture frame between 20% and 200% where 100% is considered the neutral value of the interaction. In our opinion, the *Exact Abstraction* for this scenario is `level` as it allows the specification of 100% to be the neutral value of the interactive component. 31.2% of the participants indeed selected the `level` abstraction; however, another 39.1% chose to either model the interaction as `set intensity` or as `set value`. This is not wrong but rather represents a different way of interpreting this interaction, which does not emphasize the modeling of a distinct neutral value. For the *Equalizer* scenario, where the *Exact Abstraction* is also `level` and it is more obvious that the neutral value should be explicitly modeled, agreement between participants is much higher: 70.3% of participants selected the `level` abstraction in this case.

For other categories, such as `move` (scenarios *Blind (Move)* and *RC Car*), abstracting from the scenario to the appropriate interaction specifier and especially matching the states of the interactor (up button pressed, down button pressed, no button pressed) to the corresponding actuator states (up, down, stop) was more subtle and hard to grasp for the participants. Another interesting scenario is *Blind (Position)*: here, only 23.2% of the participants selected the `position` category while 44.9% selected its parent abstraction `set value` which demonstrates that they did not include the semantic information regarding the positioning in one dimension. A possible explanation for this behavior is that scrollbar-based window blind controls are not widespread and thus were considered unnatural by the study participants.

Because our interaction abstractions are modeled on "natural" types of interaction with devices and software abstractions, we did already expect that individuals would be able to annotate devices before seeing the results of the user study. However, we were still surprised by the high accuracy and high degree of agreement with our choices, and especially by the very low amount of time that participants required to produce the descriptions, which was under one minute per scenario in most cases. Summarizing, the results of our study show that the proposed scheme is very accessible, and not only for people with good knowledge of ICT systems: most participants were able to productively use it within minutes and with only negligible prior training. Considering the timing and confidence values, the tasks were also fast and easy to perform. We expect the discrepancies between the choice of optimal and suboptimal abstractions to strongly decrease when individuals approach the task of modeling more rigorously and in the context of *actually* providing user interfaces rather than answering questions in a survey.

7.3 Assessing Usability: Deployments and Case Studies

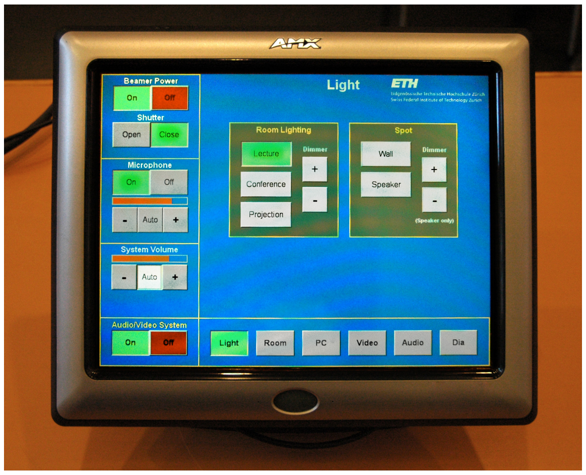
After discussing the generality of our language and its producibility for developers, we now assess the usability of interfaces that are created from our descriptions. In Section 6, we introduced a concrete implementation of a mobile application that generates interfaces for all defined interaction abstractions and enables the user to create composite, task-centric, interfaces. We want to point out that, due to our interaction descriptions being language- and device-independent, this application represents only one of several possible ways of interpreting our interaction abstractions, where the interaction abstractions were mapped to simple Android widgets or made use of the Android API for sensor access and haptic feedback. Our prototype application was tested by several members of our research group and used to control and monitor the devices described.

Test subjects reported that the generated interactors felt intuitive and appropriate for controlling and monitoring all devices. Specifically, giving only very little information (i.e., only the data type and name of the interaction abstraction without any of the optional properties) in most cases was sufficient for creating an intuitive user interface, as our descriptions consider the high-level semantics of interactions on multiple levels, as detailed in Section 4.2. Test subjects especially enjoyed those interactors that bridged multiple modalities by, for instance, making use of the sensors of the mobile device (e.g., shaking the phone or speech input). When we enriched the descriptions with some of the defined optional description elements, test subjects in particular liked the haptic feedback capabilities of the prototype application (e.g., vibrations and sounds triggered by *anchor* annotations).

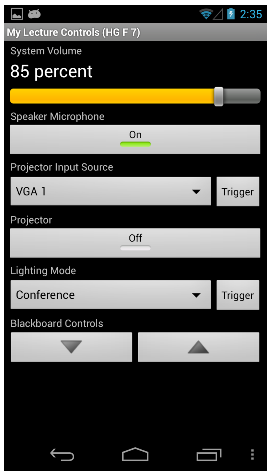
The usability of the interfaces that are generated by our specific prototype is, however, grounded in the usability of the underlying Android widgets and, therefore, does not allow to conclude that our description language necessarily leads to usable and intuitive user interfaces in all cases. Still, our prototype shows that the language can definitely be used to create good user interfaces, even though only simple mappings between our abstractions and Android widgets are employed. One could also implement device controllers that map our descriptions to different, more customized, final interfaces, or to interfaces that support even more modalities, such as gesture-based interaction. Furthermore, applications could be created that allow for more sophisticated composite user interfaces – our prototype uses rather simple widget lists for this purpose.

In the following, we discuss two case studies to illustrate different aspects of our language and the prototype application: The first targets personalized composite user interfaces in the context of a lecture hall control system. The second demonstrates that our language can indeed be used to describe devices with a complex internal state, although the interaction descriptions do not specify dependencies between different interactive components of a device.

Case Study 1: Lecture Hall Controls To test our approach within a real-world setting, we created a mockup that emulates the specific auditorium



(a)



My Lecture Controls (HG F 7)

System Volume
85 percent

Speaker Microphone
On

Projector Input Source
VGA 1
Trigger

Projector
Off

Lighting Mode
Conference
Trigger

Blackboard Controls

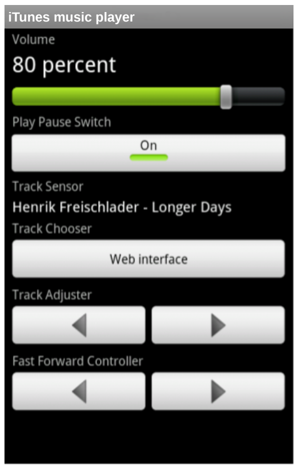
(b)

Figure 8: a) Picture of the lecture hall control system in use at our institution. b) Individually composed task-centric interface for accessing frequently used controls, rendered on an Android smartphone.

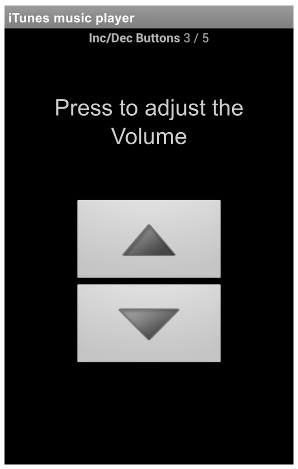
controls that are in use in our institution[5] (cf. Figure 8(a)). From past experience, we know that users who were not familiar with this system had trouble navigating its different tabs (*Lights*, *Room*, etc.) in search of their desired controls, a common mistake being that the *Video* tab was selected when looking for the controls to select the system's video input source. We also experienced that non-German-speaking individuals had trouble with some of the interface labels due to poor translation, for example with the "Beamer Power" controls (top left corner of the system in Figure 8(a); the term "Beamer" is commonly used in German to denote a "Video Projector"). Our goal was to recreate all atomic interactive components of this lecture hall control system and let users configure composite interfaces that are customized according to their individual preferences using our prototype application. We had not considered this specific auditorium control system when creating the interaction abstraction categories and designing our language.

To test our idea, we created a mockup lecture room automation back-end, added all interactive components of the auditorium control system as endpoints to that server, and described them using Microdata annotations. In total, we needed less than 30 minutes to annotate all 28 components, which include the room lighting (setting the lighting level and mode), shades and blinds controls, ventilation mode settings, and controls for the video and slide projectors, sound system, and blackboards. Some of these components have strong dependencies between each other: for instance, the projector shutter is only available when

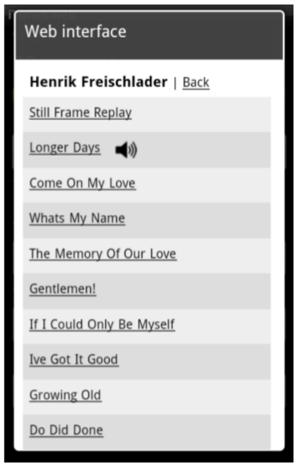
[5]An AMX LLC Level 3 Modero Auditorium Control System



(a)



(b)



(c)

Figure 9: Example user interfaces for the iTunes application, rendered on an Android smartphone: a) Composite interface that displays all interactors associated with the iTunes application. b) Full-screen user interface for the volume controller ("`move`"). c) Manually created Web interface to select tracks.

the projector itself is switched on.

The composite interface that results from combining a specific user's most frequently used controls is shown in Figure 8(b). To create that interface, one uses our prototype application to access the URL of the mockup (e.g., by scanning a barcode or an NFC tag that is attached to the physical device or to a proxy), which causes it to load all interface descriptions of the individual components and store the corresponding interfaces locally. Next, one instantiates a new task-centric interface, gives it a name (in Figure 8(b), this is "My Lecture Controls (HG F 7)"), and selects which of the 28 loaded components this composite interface should contain. The user can also choose to interact with each of the components individually by accessing the corresponding "full-screen interactor" (some examples of these were shown in Figure 5). Components can also be rearranged and can be removed from a composite interface.

Case Study 2: Music Player Our description language does not allow to explicitly model logical dependencies between interactive components of a device. This means that it cannot be used to express how different resources that all belong to the same smart thing influence each other and how they affect the global state of the device. For instance, and referring to the example shown in Section 3, switching a television set off using its on/off button has an immediate consequence on the internal state of the device (i.e., it is now switched off) and also affects its other interactors: it is not anymore possible to control the volume of the TV or change the channel. Although such dependencies cannot be modeled in our system, it can still handle complex devices whose components are tightly coupled: using our generated interfaces to interact with the iTunes application – a smart thing with strong dependencies between its actuators and sensors and a complex internal state – is intuitive and effective (see Figure 9). Although only the smart thing itself (i.e., the iTunes application) keeps the global application state and delivers only partial views to the interaction application on the smartphone, the generated interface for the music player creates the illusion of the interactor being aware of the full internal state of the iTunes application. This is possible because the interaction between the smart thing and the smartphone application is, indeed, two-way: the interaction application can permanently update the state of the rendered interactors by querying the smart thing. Therefore, if, for instance, a specific interaction becomes impossible (such as controlling the volume of the TV set in the earlier example), the user interface can immediately reflect that change. Example interactive components that we modeled for the iTunes application are its volume control (`set intensity`), Play/Pause switch (`switch`), Rewind/Fast Forward (`move`), an interface to skip tracks (`goto`), and the current track name display (`get data`).

8 Related Work

Model-based user interfaces have been investigated for a long time, initially with the goal of relieving application programmers from the task of manual GUI creation. [19] and [10] represent examples of early work that focused on the automatic generation of GUIs. This investigation led to the emergence of several model-based user interface description frameworks (see [20] for an overview and classification). However, the automatically generated interfaces were often not well adapted to the application, which led to poor user experience (see [18]) and the process of creating the models themselves proved to be rather cumbersome [21].

The Personal Universal Controller (PUC) / Pebbles project represents a pivotal step in the development of automatic user interface generation that would transform hand-held computers into universal control devices [22, 21]. There, a description concept and concrete UIDL was proposed for appliances such as televisions, telephones, VCRs, and photocopiers. The proposed UIDL focuses on enabling high-quality interfaces, where some specifications consist of as many as 100 functional elements. The authors emphasized the producibility of interface descriptions and thus have designed the language to be easy to learn and use. This has been verified in a user study where subjects needed only an hour and a half of studying a tutorial document to be able to write specifications for a VCR interface. In [23], the authors present an extension of their system and introduce "smart templates" such as *media-controls* or *time-duration*, to better encapsulate the *meaning* of interactors – the *media-controls* template, for instance, contains *Play* and *Stop* controls. In contrast to our work, appliances are thus viewed as collections of tightly coupled "appliance objects" and their descriptions explicitly include logical dependencies between these functional units. Other approaches to mitigate the problem of interface specifications that are hard to create include the development of specialized authoring software to allow developers to produce user interface descriptions, which has been done, for instance, in the MARIA project [27].

The tight coupling of functional user interface components makes interface descriptions hard to produce and, as we have shown, is not necessary to create usable and intuitive interfaces. Rather, following our approach, it is sufficient to describe the atomic interactive components of a smart thing on a level that allows to exploit metaphors that developers are already familiar with (i.e., the example symbols that are associated with our interaction abstractions). This usage of meaningful abstractions to avoid dealing with low-level details indeed represents the main motivation for model-based approaches – by focusing on making our interaction metadata simple to understand, produce, and embed, we thus attempt to "overcome the traditional separation between end users and software developers" [26]. Furthermore, in our approach, the generated interfaces are understood as a representation of the internal state of the device which is continuously updated. [11] refers to such interfaces, which feature bidirectional communication between the controlled smart thing and the interactor, as *complementary*, *duplicated*, or *detached* user interfaces, depending on what kind of (attached) user interface the controlled device itself provides. This feature directly enables to transfer interfaces from one device to another at runtime in a process called *user interface migration* [26]. Due to the platform-independence of our language, this can involve migrating user interfaces between heterogeneous devices with diverse capabilities, for instance from a smartphone to a physical switch or button. Because our language specifies the high-level semantics of concrete interactions, it also supports multiple contexts of use and preserves usability under these adaptations, properties which are referred to as *multi-targeting* and *plasticity* [3].

Our work fully integrates multimodal user interfaces for heterogeneous interactors, and thus differs from approaches such as PUC, which are aimed at "traditional" mobile user interfaces with touch panels or small keys [29]. This is important as the rise of the *ubiquitous computing* paradigm – where a person uses multiple networked computing devices that are embedded in everyday real-world objects – has led to a broadening of the design space of automatic user interface creation [28]. In the ubiquitous computing paradigm, interactions often take place in a spontaneous, ad-hoc fashion. Content and user interfaces now have to be adapted to a wide variety of devices with varying screen sizes (e.g., mobile phones vs. public displays) and heterogeneous capabilities (e.g., networked physical buttons vs. tablets with touchscreen, gyroscope, and accelerometer). This has in turn led to user interface models being increasingly abstracted from concrete GUIs to place more focus on user interaction on more abstract levels, which, in particular, targets multidevice interfaces [26, 1]. One of the first examples for this development is presented in [9], which introduces the notion of "universal interaction" and represents a first step in the development of a service architecture that supports heterogeneity in interactors and the controlled objects.

Another approach that targets the automatic provisioning of user interfaces that support multiple modalities of interaction is the XWeb project [24]. In XWeb, user interface information is conveyed using *XView* descriptions that contain information about interface elements such as icons, field names, layout, and help texts and can also be used to generate speech interfaces. In that respect, one should also mention Interplay [17], which focuses on device and content integration as well as on the interaction with devices at the level of the *task* to be accomplished. To this end, this work provides valuable insights about speech analysis to enable the system to map spoken commands to specific tasks. An ambitious project that focused on enabling people with disabilities to control everyday devices using their specialized controllers – for instance, user interfaces that are attached to wheelchairs – is the Universal Remote Console / V2 (URC) specification [31]. There, target devices (e.g., ATMs) transmit a description of their abstract input/output behavior to the controller, which then renders an appropriate user interface. Finally, the SUPPLE project [6] addresses the provisioning of alternative user interfaces by stating interface generation as a discrete constrained optimization problem that can be solved on the fly, where, for instance, a person's motor impairments are modeled as a cost function to guide the optimization. Furthermore, this article includes a discussion of some other approaches to model-based user interface generation that have been published in the last decade. In contrast to our approach, the projects mentioned in this paragraph require user interface descriptions that are difficult to create for users or, especially in the case of SUPPLE, can only be created by experts.

With respect to conceptual work in the domain of model-based user interface descriptions that studies the various abstraction levels of user interface design [27, 16], the main novel features of our interaction description language affect the level of the "Abstract User Interface[6]." At this level, the descriptions of "Abstract Interaction Objects" [16] are independent of the concrete platform and interaction modality and are, rather, described in terms of their "semantics" [26]. While Paternò and Meixner refer to the semantics of user interfaces, we consider the semantics of the interaction itself and identified three concrete components, or dimensions, of the interaction semantics that we detailed in Section 4.2. With respect to the concrete interaction abstraction categories, our language is related to the Dialog and Interface Specification Language (DISL) [8] and to XForms[7]. DISL proposes eight basic widgets for user interaction (*variablefield*, *textfield*, etc.). XForms controls include abstractions such as *trigger* (activation of a process) and *secret* (entry of sensitive information in a form). We also propose a set of basic interaction abstractions but do not mix information that relates to the type of data exchanged with the high-level semantics of an interaction (see Section 4). This abstraction and separation of concerns, combined with the bidirectional communication between interfaces and interac-

[6]http://www.w3.org/TR/abstract-ui/

[7]`http://www.w3.org/MarkUp/Forms/`

tive components, is the key to our descriptions being easy to understand and produce and still being expressive enough to cover all our considered use cases.

9 Conclusions

Most user interface description languages model user interfaces as composites of interactors like text input, value selection, and output widgets where the appropriate interactor for an interactive component is selected based on its data type. We instead propose a way to express the semantics of an interaction that enables the generation of more intuitive graphical widgets as well as the mapping of interactive components to gesture-based, speech-based, or physical interfaces. One main advantage of our approach is that the provisioning of a live interaction mechanism is reduced to the embedding of simple interaction information into the representation of a smart thing. Decomposing devices into atomic components and adding a small amount of simple information to collections of resources has proven to be well suited for describing resources in an expressive but still easy-to-use way. The high level of abstraction of this information allows for the generation of modality-independent user interfaces while taking into account the capabilities of the target device. Smart things themselves do not need to be aware of what types of devices (e.g., PCs, handheld devices, or even other smart things such as Web-enabled knobs or switches) use this information and what kinds of interfaces these provide for users to control them.

Based on this approach, we presented a taxonomy of typical high-level interaction semantics and a description scheme that allows for the automatic generation of intuitive user interfaces for smart physical things and software components. We described a mobile device controller that generates user interfaces for smart things that embed a description of their interaction semantics according to our proposed language. The evaluation of the prototype in a laboratory deployment as well as in several deployments in private homes produced good results in terms of the usability of the generated interfaces and the generality of the description language: the application can generate convenient user interfaces where the user can choose between graphical, haptic, and speech-based interfaces. Our taxonomy of interaction abstractions covers all devices (physical and virtual) that we tried to include in our deployments as well as via mockups such as a lecture theatre control system. Finally, a study of 780 participants showed that our proposed concepts for a user interface definition language can be used by tech-savvy individuals without any special training.

In conclusion, we remark that the proposed concept can be extended in several directions: First, it would be interesting to explore how context-sensitive interfaces could be better supported by tailoring interaction patterns to the user's situation. For instance, human users often wish to interact differently with a device in their immediate vicinity than when controlling the device from a remote location. Most users probably prefer gradual/relative interaction primitives over absolute interactors to interact with devices which they can physically observe (e.g., dimming the lights in the room they currently are in) while they will prefer the latter for remote control (e.g., remote-controlling the lighting in their home from abroad). Information about a user's location relative to the interactive component could thus be considered when rendering the user interface. Second, our approach of modeling interaction semantics could be applied not only to interactive components but also to interactors like physical buttons or software primitives. For instance, a knob could embed a description indicating that it is usable for controlling any interactive component of type `set value`. Given this information, one can envision the user-driven or even automatic matching of interactive components and interactors in flexible smart environments. If desired by users, physical interactors like switches could easily be assigned to control any device that has appropriate interaction semantics and would not be anymore limited to controlling statically assigned interactive components.

References

[1] Beaudouin-Lafon, M. Designing Interaction, not Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (New York, NY, USA, 2004), M. F. Costabile, Ed., ACM, pp. 15–22.

[2] Beigl, M., Schmidt, A., Lauff, M., and Gellersen, H.-W. The UbicompBrowser. In *Proceedings of the 4th ERCIM Workshop on User Interfaces for All* (Oct. 1998), C. Stephanidis and A. Waern, Eds., pp. 51–86.

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers 15*, 3 (2003), 289–308.

[4] Corcoran, P. M., and Desbonnet, J. Browser-style Interfaces to a Home Automation Network. *IEEE Transactions on Consumer Electronics 43*, 4 (1997), 1063–1069.

[5] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[6] Gajos, K. Z., Weld, D. S., and Wobbrock, J. O. Automatically Generating Personalized User Interfaces with SUPPLE. *Artificial Intelligence 174*, 12/13 (2010), 910–950.

[7] Guinard, D., Trifa, V., Mattern, F., and Wilde, E. From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices. In *Architecting the Internet of Things*, D. Uckelmann, M. Harrison, and F. Michahelles, Eds. Springer, Berlin, Germany, 2011, pp. 97–129.

[8] Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. Human-Centered Engineering of Interactive Systems with the User Interface Markup Language. In *Human-Centered Software Engineering – Software Engineering Models, Patterns and Architectures for HCI*, A. Seffah, J. Vanderdonckt, and M. C. Desmarais, Eds., Human-Computer Interaction Series. Springer, Berlin, Germany, 2009, pp. 139–171.

[9] Hodes, T. D., Katz, R. H., Servan-Schreiber, E., and Rowe, L. Composable Ad-hoc Mobile Services for Universal Interaction. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Networking and Computing* (New York, NY, USA, Sept. 1997), L. Pap, K. Sohraby, D. B. Johnson, and C. Rose, Eds., ACM, pp. 1–12.

[10] Kim, W. C., and Foley, J. D. Providing High-level Control and Expert Assistance in the User Interface Presentation Design. In *Proceedings of the Human-Computer Interaction, INTERACT '93, IFIP TC13 International Conference on Human-Computer Interaction* (New York, NY, USA, Apr. 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. N. White, Eds., ACM, pp. 430–437.

[11] Lorenz, A. Architectural patterns for applications with external user interface elements. *Pervasive and Mobile Computing 9*, 2 (2013), 269–280.

[12] Mathieson, K., Peacock, E., and Chin, W. W. Extending the Technology Acceptance Model: The Influence of Perceived User Resources. *ACM SIGMIS Database 32* (2001), 86–112.

[13] Mattern, F., and Floerkemeier, C. From the Internet of Computers to the Internet of Things. In *From Active Data Management to Event-Based Systems and More*, K. Sachs, I. Petrov, and P. Guerrero, Eds., vol. 6462 of *LNCS*. Springer, Berlin, Germany, 2010, pp. 242–259.

[14] Mayer, S., and Guinard, D. An Extensible Discovery Service for Smart Things. In *Proceedings of the 2nd International Workshop on the Web of Things* (New York, NY, USA, June 2011), D. Guinard, V. Trifa, and E. Wilde, Eds., ACM.

[15] Mayer, S., Guinard, D., and Trifa, V. Facilitating the Integration and Interaction of Real-World Services for the Web of Things. In *UrbanIOT 2010; Workshop at the Internet of Things 2010 Conference (IoT 2010)* (Tokyo, Japan, Nov. 2010).

[16] Meixner, G., Paternò, F., and Vanderdonckt, J. Past, Present, and Future of Model-Based User Interface Development. *i-com 10*, 3 (2011), 2–11.

[17] Messer, A., Kunjithapatham, A., Sheshagiri, M., Song, H., Kumar, P., Nguyen, P., and Yi, K. H. InterPlay: A Middleware for Seamless Device Integration and Task Orchestration in a Networked Home. In

Proceedings of the 4th IEEE International Conference on Pervasive Computing and Communications (Washington, D.C., USA, Mar. 2006), IEEE Computer Society, pp. 296–307.

[18] Myers, B., Hudson, S. E., and Pausch, R. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction 7*, 1 (2000), 3–28.

[19] Myers, B. A. A New Model for Handling Input. *ACM Transactions on Information Systems 8*, 3 (1990), 289–320.

[20] Navarre, D., Palanque, P., Ladry, J.-F., and Barboni, E. ICOs: A Model-Based User Interface Description Technique Dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. *ACM Transactions on Computer-Human Interaction 16*, 4 (2009).

[21] Nichols, J., and Myers, B. A. Creating a Lightweight User Interface Description Language: An Overview and Analysis of the Personal Universal Controller Project. *ACM Transactions on Computer-Human Interaction 16*, 4 (Nov. 2009).

[22] Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., and Pignol, M. Generating Remote Control Interfaces for Complex Appliances. In *Proceedings of the 2002 International Conference on Intelligent User Interfaces* (New York, NY, USA, Jan. 2002), ACM, pp. 161–170.

[23] Nichols, J., Myers, B. A., and Litwack, K. Improving Automatic Interface Generation with Smart Templates. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces* (New York, NY, USA, Jan. 2004), J. Vanderdonckt, N. J. Nunes, and C. Rich, Eds., ACM, pp. 286–288.

[24] Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P. Cross-modal Interaction using XWeb. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces* (New York, NY, USA, Jan. 2000), ACM, pp. 191–200.

[25] Ostermaier, B., Kovatsch, M., and Santini, S. Connecting Things to the Web using Programmable Low-power WiFi Modules. In *Proceedings of the 2nd International Workshop on the Web of Things* (New York, NY, USA, June 2011), D. Guinard, V. Trifa, and E. Wilde, Eds., ACM.

[26] Paternò, F. Model-based tools for pervasive usability. *Interacting with Computers 17*, 3 (2005), 291–315.

[27] Paternò, F., Santoro, C., and Spano, L. D. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction 16*, 4 (Nov. 2009).

[28] Ponnekanti, S. R., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Proceedings of the 3rd International Conference on Ubiquitous Computing* (Berlin, Germany, Sept. 2001), G. D. Abowd, B. Brumitt, and S. A. Shafer, Eds., vol. 2201 of *LNCS*, Springer, pp. 56–75.

[29] Tokunaga, E., Kimura, H., Kobayashi, N., and Nakajima, T. Virtual Tangible Widgets: Seamless Universal Interaction with Personal Sensing Devices. In *Proceedings of the 7th International Conference on Multimodal Interfaces* (New York, NY, USA, Oct. 2005), G. Lazzari, F. Pianesi, J. L. Crowley, K. Mase, and S. L. Oviatt, Eds., ACM, pp. 325–332.

[30] Weiss, M., and Guinard, D. Increasing Energy Awareness Through Web-enabled Power Outlets. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia* (New York, NY, USA, Dec. 2010), M. C. Angelides, L. Lambrinos, M. Rohs, and E. Rukzio, Eds., ACM.

[31] Zimmermann, G., Vanderheiden, G., and Gilman, A. Prototype Implementations for a Universal Remote Console Specification. In *Extended Abstracts of the 2002 Conference on Human Factors in Computing Systems* (Apr. 2002), L. G. Terveen and D. R. Wixon, Eds., ACM, pp. 510–511.