Interaction Primitives: Describing Interaction Capabilities of Smart Objects in Ubiquitous Computing Environments

Bram J.J. van der Vlist, Gerrit Niezen, Jun Hu and Loe M.G. Feijs
Department of Industrial Design
Technische Universiteit Eindhoven
Eindhoven, Netherlands
{b.j.j.v.d.vlist, g.niezen, j.hu, l.m.g.feijs}@tue.nl

***Abstract*—The design of ubiquitous computing environments introduces challenges on both infrastructure level and the level of interaction design. To support the transition from a device-oriented paradigm towards a system-oriented paradigm with increased interoperability, we need a framework to describe interactions and interactive objects in such a way that the physical and contextual meaning of the interaction is preserved. In this paper we describe a way to model interactions in terms of their essential elements, which we call *interaction primitives*: The smallest addressable interaction elements that have a meaningful relation to the interaction itself. By semantically describing the user interaction capabilities of devices which are meaningful to be shared with other devices, interoperability will not only become possible at infrastructure level, but may also improve user interaction in ubicomp environments.**

***Index Terms*—ontologies; user interaction; interoperability; semantic mapping; interaction primitives**

I. Introduction

This paper introduces a view on user interaction in ubiquitous computing environments [1], where user interaction is no longer limited to “one user - one product”, but where users interact with an ecosystem of interconnected devices.

To support this shift in user interaction from a device-centred approach to a system-centred approach, we need the ability to describe interactions and objects for interaction in such a way that the physical and contextual meaning of the interaction is preserved. In this paper we describe a framework to model the essential elements of an interaction, which we call *interaction primitives*, the smallest addressable interaction element that has a meaningful relation to the interaction itself. We hypothesize that user intentions can be inferred when the interaction capabilities (in addition to device capabilities in general) of devices are described semantically, and the relationships between the devices or the interaction elements of the devices are described using *semantic connections* [2].

We model these interactions in a mutually understandable way by describing them in an ontology. The actual ontology is described in section V, and we explain the use of the ontology with a volume control example.

One possible solution to solve the interoperability problem at the infrastructure level is the software platform developed within the SOFIA[1] (Smart Objects For Intelligent Applications) project. SOFIA is a European research project within the ARTEMIS framework that attempts to make information in the physical world available for smart services—connecting the physical world with the information world. The goal of the SOFIA IOP (Interoperability Platform) is that devices will be able to interact on a semantic level, utilizing (potentially different) existing underlying services or service architectures. Part of this effort is to define a core ontology that describes commonly used concepts, and also model exemplary domains, more completely in a formal ontology that is expressed in OWL[2] (Web Ontology Language).

The use of interaction primitives to model the essential elements of user interaction as is introduced in this paper, is built on the SOFIA IOP, and is currently being implemented by various project partners. Some of the concepts are planned to become part of the SOFIA core ontology, while others will be developed into application- or domain-specific ontologies.

II. Related Work

In order to describe devices in terms of their physical interaction capabilities, we started off with investigating various taxonomies related to defining and classifying input devices. Foley et al. [3] describe a taxonomy of input devices that are structured according the graphic subtasks they can perform: position, orientation, select, path, quantify and text entry. Card et al. [4] pointed out that the Foley taxonomy has not tried to define a notion of completeness, and is thus not generic enough. Single devices appear many times in the levels of the tree, which makes it difficult to understand the similarities among devices. In this sense, Buxton’s taxonomy [5] is a major step forward, identifying the elementary properties of position, motion, and pressure. MacKinlay, Card and Robertson [6] extended Buxton’s work to propose additional physical properties that underly most devices. They follow mappings from the raw physical transducers of an input device into the semantics of the application. Our work builds on this concept, and is described in more detail in the following sections.

[1]http://www.sofia-project.eu/
[2]http://www.w3.org/TR/owl2-overview/

MacKinlay et al. consider the following to be important parts of an input device:

- the geometry of the transducers of physical manipulation, e.g. rotation around the Z-axis;
- the domain of values that the transducer can produce, e.g. $0° - 270°$;
- device resolution, e.g. maps continuous region into $\{0°, 45°, 90°\}$;
- connections among devices, e.g. mapping `SelectorKnob` to `AMTuner` and `FMTuner`, and mapping the rotation of $\{0°, 45°, 90°\}$ to the set {`OFF`, `ON`} for both tuners.

They then define an input device to be a 6-tuple `<M, In, S, R, Out, W>` where:

- `M` is a manipulation operator, corresponding to the physical property vector;
- `In` is the input domain set over which the manipulation operator will sense a value;
- `S` is the current state of the device;
- `R` is a resolution function that maps from the input domain to the output domain set;
- `Out` is the output domain set, which describes the range of the resolution function; and
- `W` is a general purpose set of device properties that describes additional aspects of how the input device works, such as its physical characteristics or its internal mechanism.

To describe the connections between input devices and application parameters, the output domain set of one device is mapped to the input domain set of another device, typically an output device. Only three parameters are needed: The output domain of the first device, the mapping function and the input domain of the second device, e.g. *Connect* (`VolumeKnob, Volume,` $f(\theta\ degrees) = C_v \times \theta\ decibels$) where C_v is a constant of proportionality, determined by the gain of the control and conversion factors among the units of measurement. In the following sections, we build on these concepts to develop a generic model for describing user interaction in a ubiquitous computing environment.

Ontologies lend themselves well to describing the characteristics of devices, the means to access such devices, and other technical constraints and requirements that affect incorporating a device into a smart environment [7]. Current RDF-based schemas for representing information about device characteristics (namely W3C's CC/PP and WAP Forum's UAProf) directly relate to this.

The User Agent Profile (UAProf) specification, used to describe the capabilities of mobile devices, distinguishes between hardware and software components for devices, but the descriptions of interaction capabilities are very limited. For example, in the Nokia 5800 XpressMusic UAProf profile[3], the only descriptions of user interaction capabilities are `PhoneKeyPad` as `Keyboard`, and "2" as `NumberOfSoftKeys`. Other user interaction capabilities are defined in a Boolean fashion of yes/no, e.g. `SoundOutputCapable`, `TextInputCapable`, `VoiceInputCapable`.

A number of ontologies have been developed for ubiquitous computing environments that may potentially be used to describe device capabilities and characteristics. Chen et al. [8] defined SOUPA, a context ontology to support ubiquitous agents in their Context Broker Architecture (CoBrA). The ontology may be used to model devices on a very basic level (e.g. typical object properties are `bluetoothMAC` or `modelNumber`), but it has no explicit support for modeling user interaction and more general device capabilities.

The SPICE Mobile Ontology[4] allows for the definition of device capabilities in a sub-ontology called Distributed Communication Sphere (DCS) [9]. A distinction is made between device capabilities, modality capabilities and network capabilities. It is assumed that the properties that are used to describe outputs (e.g. acoustic/visual/tactile) can also be used to describe inputs. This is not always the case, as for inputs we would rather use physical properties like position, movement, rotation, force and torque. Also, the properties should not be used to describe the content that may be exchanged, but the actual interaction capabilities. As an example, if a device has an `AcousticOutputModalityCapability`, it should mean that the device can provide user feedback (e.g. in the form of computer-generated speech or an audible click) and not that the device is capable of playing music.

III. User interface model

To enable user interaction in smart spaces on the level that was sketched in the introduction, the developer community needs to agree on a way of describing the various elements involved in the interactions. These interaction elements or controls are physical by nature (i.e. they are material parts or at least directly perceivable in the physical world), which means that their physical meaning and some of physical properties need to be preserved while describing them. Later in section IV a simple example will be described, explaining why the previous statement is important, especially when considering that this user interaction data is shared and used by other devices.

Figure 1 shows our proposal to model user interfaces in terms of their physical, real-world interaction properties (like position, movement, rotation, force and torque) and their transformation towards the consequences they have in the digital domain (e.g. triggering interaction events, changing states). The entities represented in the model as circles are what we call *interaction primitives*, the smallest addressable interaction element that has a meaningful relation to the interaction itself.

On the left-hand side of the figure we plot entities that sense *physical* properties like position, movement or pressure. We consider these properties to be very *generic*, as they do not report a user's intention directly. The inputs first need to be

[3]nds1.nds.nokia.com/uaprof/Nokia5800d-1r100-2G.xml

[4]http://ontology.ist-spice.org/

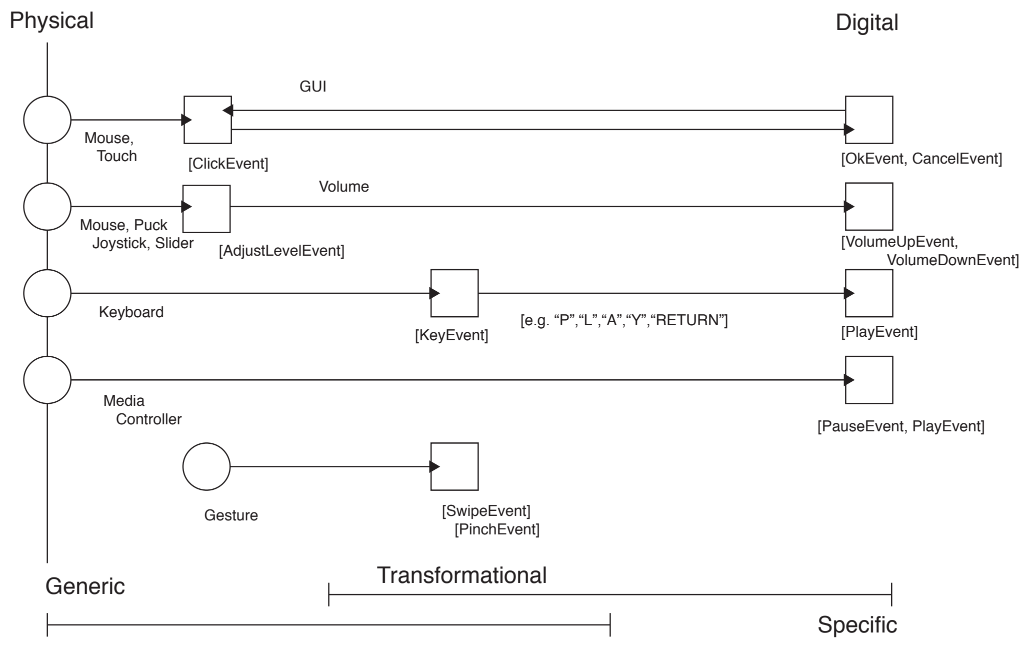

Fig. 1. User Interface Model

transformed into an intentional event (events that express user intention). This can happen directly, for example pressing a play button, which is transformed into an `PlayEvent`. It can also follow a series of intermediate transformational steps, where a sequence of interaction events (possibly happening on different devices) may be used to capture the user's intent. This sequence of events is then transformed into a single intentional event.

On the right-hand side of the figure we have the *digital* entities that represent the intentional events. We consider these entities to be very *specific*, as they communicate the (assumed) intention of the user's actions directly.

Entities and their relationships in an interaction together form an *interaction path*. The interaction exchange or action between elements in the path is conducted via one or more *interaction channels* along which information or action is communicated [10]. As an example, a typical interaction path in Figure 1 would be:

`Keyboard` → `KeyEvent` → `PlayEvent`

while an interaction channel exists between `Keyboard` and `KeyEvent`.

During the transformation from physical to digital, the interaction devices (or their interaction primitives) also move from generic to more specific, where generic user interfaces start off very generic and stay generic or transformational (meaning they have been transformed but still need further transformation). This means that such interaction devices or interaction primitives can still be transformed into many different events or states. When an interaction primitive travels from generic to specific with a single transformation (like the media controller buttons) it means that that these interaction primitives are very specific UI elements (i.e. have one single function). An example of such an interaction primitive would be a hardware button with a specific label that is only used for one function. As another example, consider a gesture; i.e., a (less) generic interaction primitive that transforms from a physical movement that is sensed in a certain way, to the digital representation of that gesture, being a "pinch", "swipe" etc. The pinch and swipe are still considered transformational because they still need to be transformed further to result in a certain interaction event. However, in the initial transformation, some meaning is preserved (i.e. the physical characteristics of the gesture). These characteristics limit the number of actual events the gesture can still be transformed into, e.g. a "swipe right" gesture should not be transformed into a "navigate forward" action, as this is the way we usually navigate backwards. Table I shows some of the possible transformational events we consider to be applicable to smart home environments in which multimedia and lighting devices are connected for certain applications.

Event	Entity this event can be performed on
AdjustLevel	Volume, Lighting
switchOnOff	Lighting, any SmartObject
Navigate	Playlist, Menu, SequentialData
Undo/Redo	Any interaction event
Stop/Start	Application, Media
DragAndDrop	Media
Query	Media, other events

TABLE I
EXAMPLES OF TRANSFORMATIONAL EVENTS IN A SMART ENVIRONMENT

Although describing user interaction capabilities of devices according to the user interface model is valid for user interaction in general, it is specifically relevant when we consider the notion of a smart space through which this interaction data can be exchanged. To achieve this, all events that need to be shared must be modelled in a mutually understandable way. A good way of modelling them would be an ontology, as is shown in section V.

When modelling, only that which is meaningful to be shared with other devices is considered. It is not necessary to describe interactions that are internal to the device and that are not shared. An accelerometer, for example, may be modelled as a separate device, sharing the raw accelerometer data to be used by other devices. However, when integrated into smart phones, the accelerometer's data can often be abstracted as part of an interaction path, e.g. to only share the orientation of the device, or specific gestures measured with the accelerometer. In this case, the raw values may only need to be available locally on the device, to be used by the developers of other device-specific applications.

IV. Example: Volume control problem

To underline the relevance of describing a device's interaction capabilities according to the user interface model we proposed, let us consider an example. We take a fairly simple example of sharing a "rocker switch" or a group of two buttons that can often be found on smart phones or media players to control the volume of various (local) audio sources (e.g. music, ringer, movie, etc.). It is likely to assume that, when considering a scenario where many devices are interconnected and user interaction information can be shared, controlling the volume of music playing remotely with the rocker switch on your smart phone will be desirable.

Rocker switches to control volume come in different versions, but to keep it simple, we consider rocker switches that are labeled with (+) and (-) (like we find on the iPhone 4) and versions that are not labelled (like we find on the Samsung Galaxy S and iPhone 3G(S)). The way the labeled buttons appear in the physical world, prescribe that the part labeled (+) should be mapped to `adjustLevel` "up" or directly to `VolumeLevelUpEvent`, and `adjustLevel` "down" or `VolumeLevelDownEvent` for the part labeled (-). For the `AdjustLevel` transformation also see table I.

For the unlabeled versions it is more difficult, as just choosing an arbitrary mapping might result in mappings that are not expected by a user. If we take for instance the unlabeled rocker switch of the Samsung Galaxy S, we find that they mapped the top part of the rocker switch to "volume up" and the lower part to "volume down" (when holding the phone in upright/portrait position; Figure 2a). When using the phone in landscape position, the right button is mapped to "volume up", and the left to "volume down", which still makes sense (Figure 2b). When rotated 180 degrees, from both portrait and landscape position, the chosen mapping becomes a little confusing as the mapping now seems reversed (Figure 2c/d). Now, suddenly, the mapping that was chosen appears less natural.

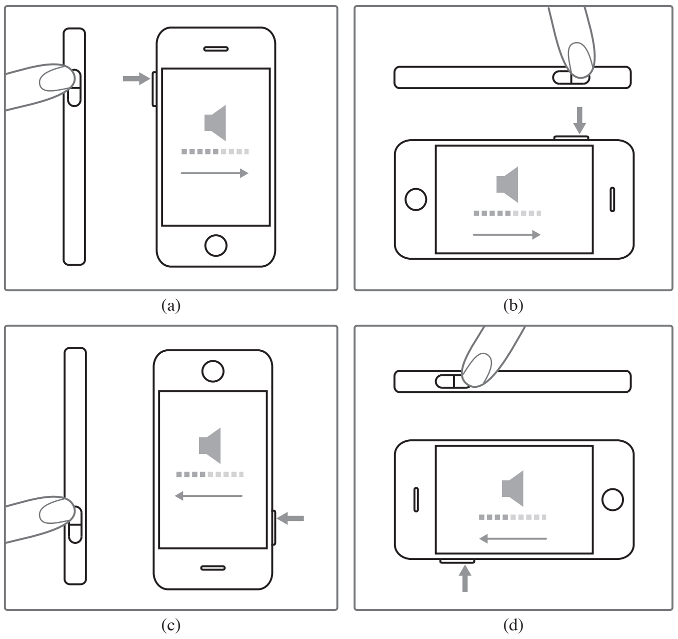
Fig. 2. Volume adjust problem: (a) portrait position, volume up; (b) landscape position volume up; (c) portrait position (rotated 180°), volume down; (d) landscape position (rotated 180°), volume down.

With something as trivial as controlling the volume, this is not much of a problem. But when the concepts behind the mappings become more complicated, issues like those just described become more problematic.

This simple example shows the importance of *semantic mapping*, where leaving out physical properties that are meaningful to a user may result in unexpected behaviour. What we would like to propose as an interaction primitive for an unlabelled rocker switch, is thus not only mapping the rocker switch up and down positions, to volume up and down, but also taking the orientation into account. Thus making the mapping context (orientation) dependent, and sharing the more meaningful, contextualized mapping.

V. Semantic interaction ontology

The Semantic Interaction ontology we have developed is shown in Figure 3. As an example of how the ontology may be used, we start off by defining a smart object and its interaction primitives. Recalling that it is only necessary to describe interaction primitives of a device if we use that device's interaction primitive to control another device through the smart space, we can describe the volume control rocker switch on a smart phone as an interaction primitive:

```
SmartPhone rdf:type SmartObject
PhoneRockerSwitch rdf:type InteractionPrimitive
SmartPhone hasInteractionPrimitive PhoneRockerSwitch
```

We now need to define the properties of the interaction primitive. We start by describing the range measure, or the range of values that the interaction primitive can produce (e.g. the rocker switch can produce `Up`, `Down` or `Neutral` values).

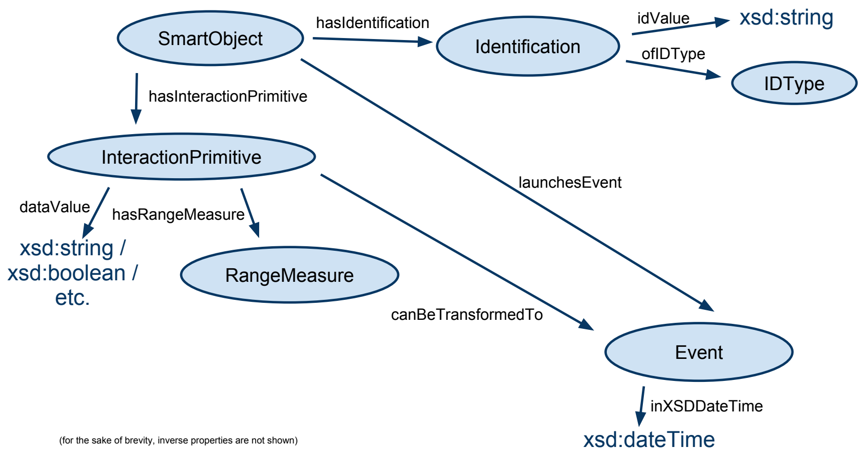

Fig. 3. Semantic Interaction Ontology

Range measure	Possible values
Binary	True/False, 0 or 1
SingleDigit	up to 9 discrete values
DoubleDigit	up to 99 discrete values
TripleDigit	up to 999 discrete values
LargeDigit	more than 1000 discrete values

TABLE II
RANGE MEASURES FOR INTERACTION PRIMITIVES

These range measures are similar to the measure of the domain set used by MacKinlay (described in section II). Using the range measures, we can then infer which transformational events may be used to map the input values to other interaction primitives or events. In the next version of the ontology, it should be possible to also describe the different manipulation operators of the interaction primitive, e.g. rotation on the z-axis or movement along the y-axis.

Table II shows the current range measures defined in the ontology. In our example we specify the `RangeMeasure` of our interaction primitive as follows:

```
PhoneRockerSwitch hasRangeMeasure SingleDigit
```

The actual data value of the interaction primitive is described using the `dataValue` property. Data values may be strings, boolean values or other datatypes, e.g.:

```
PhoneRockerSwitch dataValue "neutral"^^xsd:string
```

When `PhoneRockerSwitch` is pressed, the data value is updated with:

```
PhoneRockerSwitch dataValue "up"^^xsd:string
```

This enables other devices to make use of the user input on the `PhoneRockerSwitch`, irrespective of the interaction events generated. In fact, using `Transformation`, it becomes possible to map the physical, generic button presses from interaction primitives like `PhoneRockerSwitch` to specific high-level events like `VolumeUpEvent` or `VolumeDownEvent` using the default transformation `AdjustLevel` as is described in table I.

By specifying the transformation using the proper OWL 2 DL (Description Logic) semantics, the reasoner should be able to infer which user inputs can be mapped to which specific high-level events. This shows up as a `canBeTransformedTo` property between an interaction primitive and an event.

In our example, this means that the following relationship will be inferred:

```
PhoneRockerSwitch canBeTransformedTo VolumeEvent
```

where the `"up"` data value may then be mapped to `VolumeUpEvent` and the `"down"` may be mapped to `VolumeDownEvent`, which are both sub-classed from `VolumeEvent`. This prevents situations (as described in section IV) where arbitrary mappings causes some of the semantics of the interaction to disappear.

VI. CONCLUSION

Even though the Semantic Interaction Ontology describes parts of a `SmartObject`, it does not fully describe all the properties and capabilities of the smart object. It only describes its interaction related properties. Particularly it defines the `SmartObject` *interaction primitives* and its *identification* means.

Idempotency is the property of being able to perform the same action twice or more times in sequence, and end up with the same result as if the action was performed once. In the triple store, defining a smart object or interaction primitive is idempotent as long as the definition does not change on the triple-level. The idempotency of interaction events depends on whether a new timestamp is used when inserting the event into the triple store.

The ontology supports the description of interaction data generated by interaction devices and sensors. Additionally, it shows that an interaction primitive may trigger an interaction event or a state change that may need to be specified in more detail by a more application-specific ontology. That is to say, this ontology may also be used to perform semantic mapping from the interaction data to user goals and/or available services [11]. Any additional information related to the smart object may be added by extending the schema defined in the Semantic Interaction Ontology.

The ontology opens up the way to context-based interaction device reconfiguration. For example, if a Context Monitor application recognizes a situation where the `PhoneRockerSwitch` should no longer control the volume, but adjust the level of lighting instead, the triple could be modified accordingly. Just such a simple change would implement a behaviour that adapts to the situation.

Context-dependent functionality changes of a control may not necessarily be a desirable feature and there is a long standing discussion in user interface research on whether or not to allow for such behaviour. It should however be noted that we only consider context-dependent meaning change with generic interaction primitives, that in itself do not have a specific, function related meaning (and might already being used for different functions, like the rocker switch in the example). Additionally, the re-mapping is only considered for those interaction elements with compatible transformational properties, e.g. the rocker switch may only be mapped to other `AdjustLevel` transformations, and not to `Start/Stop`. The specified range measures are used to control the re-mapping between a interaction primitive and an interaction event, in a similar way that the input and output domains of [6] are used to control the expressiveness between an input device and its application parameter.

Besides automatic context-dependent functionality changes of controls, we especially consider user-initiated re-mapping of controls. By enabling users to make associations, or *semantic connections* [2] between devices or interaction elements and devices, users can express their *intentions* in terms of mapping controls [11]. These *semantic connections*, together with context information and user interaction data described according to the model we proposed, may be a good start to enabling semantic interaction in ubiquitous computing environments.

While many of the issues discussed in this paper apply to user interaction in general, the way in which interaction events and interaction primitives are distributed between multiple devices makes our work especially applicable to ubiquitous computing environments. Semantic mappings between interaction primitives and interaction events happen not only on a single device, and mappings between different devices is also supported.

The use of interaction primitives to model the essential elements of user interaction is currently being implemented by various project partners, and up until now seem promising. A thorough evaluation and validation is part of our next steps, as well as integration of the introduced concepts into the SOFIA core ontology or application- or domain-specific ontologies.

ACKNOWLEDGMENT

SOFIA is funded by the European Artemis programme under the subprogramme SP3 Smart environments and scalable digital service.

REFERENCES

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, September 1991.

[2] B. van der Vlist, G. Niezen, J. Hu, and L. Feijs, "Semantic connections: Exploring and manipulating connections in smart spaces," in *Computers and Communications (ISCC), 2010 IEEE Symposium on*, 22-25 2010, pp. 1 –4.

[3] J. D. Foley, V. L. Wallace, and P. Chan, "The human factors of computer graphics interaction techniques," *IEEE Computer Graphics and Applications*, vol. 4, no. 11, pp. 13–48, 1984.

[4] S. K. Card, J. D. Mackinlay, and G. G. Robertson, "A morphological analysis of the design space of input devices," *ACM Transactions on Information Systems (TOIS)*, vol. 9, no. 2, p. 99, 1991. [Online]. Available: http://portal.acm.org/citation.cfm?id=123078.128726

[5] W. Buxton, "Lexical and pragmatic considerations of input structures," *ACM SIGGRAPH Computer Graphics*, vol. 17, no. 1, pp. 31–37, 1983. [Online]. Available: http://portal.acm.org/citation.cfm?id=988586

[6] J. Mackinlay, S. Card, and G. Robertson, "A Semantic Analysis of the Design Space of Input Devices," *Human-Computer Interaction*, vol. 5, no. 2, pp. 145–190, Jun. 1990.

[7] "OWL Web Ontology Language Use Cases and Requirements," http://www.w3.org/TR/webont-req/, 2004.

[8] H. Chen, F. Perich, T. Finin, and A. Joshi, "SOUPA: standard ontology for ubiquitous and pervasive applications," in *Mobile and Ubiquitous Systems: Networking and Services, MOBIQUITOUS 2004*, 2004, pp. 258–267.

[9] C. Villalonga, M. Strohbach, N. Snoeck, M. Sutterer, M. Belaunde, E. Kovacs, A. Zhdanova, L. Goix, and O. Droegehorn, "Mobile ontology: Towards a standardized semantic model for the mobile domain," in *Service-Oriented Computing-ICSOC 2007 Workshops*. Springer, 2009, pp. 248–257. [Online]. Available: http://www.springerlink.com/index/r5410m517j67w455.pdf

[10] E. Dubois and P. Gray, "A design-oriented information-flow refinement of the asur interaction model," *Engineering Interactive Systems*, vol. 4940/2008, pp. 465–482, 2008.

[11] G. Niezen, B. van der Vlist, J. Hu, and L. Feijs, "From events to goals: Supporting semantic interaction in smart environments," in *Computers and Communications (ISCC), 2010 IEEE Symposium on*, 22-25 2010, pp. 1029 –1034.