Part I

FRAMING THE PROBLEM AND CURRENT STATE-OF-THE-ART

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* anglo-romanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

Part II

DESIGN ITERATIONS AND CONSTRUCTING A THEORY

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* anglo-romanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

Part III

CONTRIBUTIONS AND EVALUATION

In this part of the thesis, the more general concepts and techniques that can be applied to ubiquitous computing are described. These concepts and techniques were extracted from work done during the three design iterations.

8

EVENT MODELLING

[Artefacts] mediate activity that connects a person not only with the world of objects, but also with other people. This means that a person's activity assimilates the experience of humanity.

— Aleksei N. Leontiev, founder of activity theory

Parts of this chapter appear in [59].

Events are notable occurrences that can be associated with people, places and objects at a specific time instant, or during a specific time interval. In the previous chapter the modelling of objects and their capabilities was described. The focus of this chapter is on how the event and its associated time instant/interval can be modelled. The modelling of people and places is considered to be outside the scope of this thesis.

Interaction events were first introduced in Section 3.2.

Interaction events are generated when a user interacts with a smart object. Interaction events are high-level input events which report the intention of the user's action directly, rather than just reporting the associated hardware input event that triggered the action. This high level of abstraction enables developers to write applications which will work across different devices and services, without having to write specific code for each possible input device.

The W3C Web Events Working Group defined four conceptual layers for interactions, in the context of touch- and pen-tablet interaction [1]:

PHYSICAL This is the lowest layer, and deals with the physical actions that a user takes when interacting with a device, such as pressing a physical button.

GESTURAL This is layer describes mappings between the lower and upper layers; for example, a "pinch" gesture may represent the user placing two fingers on a screen and moving them together at the physical layer. This may map to a "zoom-in" event at the representational layer.

REPRESENTATIONAL This layer indicates the means by which the user is performing a task, such as zooming in, panning, navigating to the next page, activating a control, etc.

INTENTIONAL This layer indicates the intention of the task a user is trying to perform, such as viewing more or less detail (zooming in and out), viewing another part of the larger picture (panning), and so forth.

Interaction Event	Entity this event can be performed on
AdjustLevelEvent	Volume, Lighting
switchOnEvent	Lighting, any SmartObject
NavigateEvent	Playlist, Menu, SequentialData
UndoEvent	Any other interaction event
StopEvent	Application, Media

Table 6: Examples of interaction events in a smart environment

Interaction events can be defined at three of the layers, with the exception of the intentional layer. In Table 6 examples of possible interaction events are shown, together with possible entities associated with these events. Most of these interaction events exist at the representational layer, which are events that have significant meaning.

These events all occur at a specific time or during a specific time interval. In Web Ontology Language (OWL), there are are two approaches to modelling time:

- Using datatype properties – event instances can be related to a literal with a XML Schema Definition (XSD) datatype such as `xsd:date` or `xsd:dateTime`.
- Using object properties – classes are used to define temporal intervals, and event instances are linked to instances of these classes using object properties.

The advantage of linking event instances directly with dates is simplicity. There are fewer abstractions to deal with, and it is easier to sort events chronologically and compare them [80]. On the other hand, working with temporal intervals provide more flexibility and allows for more detailed temporal reasoning.

8.1 RELATED WORK

We now look at various existing event ontologies that we build upon to model interaction events in ubiquitous computing environments. We will also look at how temporal reasoning is performed with ontologies.

8.1.1 *The Event Ontology*

The Event Ontology (EO)[1] was developed within the context of the Music ontology[2] at Queen Mary, University of London. Although

1 http://motools.sf.net/event/event.html
2 http://purl.org/ontology/mo/

originally created to describe musical performances and events, it is currently the most commonly used event ontology in the Linked Data community [80].

The Timeline[3] ontology, used to define time instants and intervals, also forms part of this collection of ontologies. Reasoning with temporal information is discussed further in Section 8.1.5.

8.1.2 *DUL*

The DOLCE+DnS UltraLight (DUL) upper ontology is a lightweight version of the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) ontology. DUL defines the class `Event` next to the disjoint upper classes `Object`, `Abstract` and `Quality` [77]. DUL allows for both the approaches to modelling time with OWL, either with the `hasEventDate` datatype property, or with a `TimeInterval` class and the `isObservableAt` object property.

Events can be related to a `Place` with the `hasLocation` property. Alternatively, events can be related to a `SpaceRegion` with the `hasRegion` property, where `SpaceRegion` resolves to a geospatial coordinate system. DUL uses a `hasParticipant` property to relate an event to an object, and uses the `hasPart` property to link events to sub-events.

8.1.3 *Event-Model-F*

The Event-Model-F ontology extends the DUL ontology to describe events in more detail. To describe the participation of an object in an event, the Event-Model-F ontology uses the Descriptions and Situations (DnS) ontology design pattern [80].

The DnS pattern is described in more detail in Chapter 9.

An object is defined as a `Participant`, where `LocationParameter` is used to describe the general spatial region of the object [77]. `TimeParameter` describes the general temporal region when the event happened by parametrizing a DUL `TimeInterval`. A composite event `Composite` is composed out of a number of `Components`.

8.1.4 *Linked Open Descriptions of Events (LODE)*

The Linked Open Descriptions of Events (LODE) ontology[4] is an ontology for publishing descriptions of historical events as Linked Data. It builds upon the work of the previous ontologies described in this section, in order to improve interoperability with legacy event collections. Its `Event` class is directly equivalent to those defined by EO and DUL.

3 http://motools.sf.net/timeline/timeline.html
4 http://linkedevents.org/ontology/

It uses time intervals to link events to ranges of time, where its `atTime` property is a sub-property of the DUL `isObservableAt` property. There is also a distinction between places and spaces, where the `inSpace` property relates the event to a space, and the `atPlace` property is a sub-property of the DUL `hasLocation` property.

8.1.5 *Ontologies for temporal reasoning*

Temporal reasoning is used when working with time intervals, for example when using Allen's Interval Algebra to define temporal relations between events. There are 13 base relations in this algebra, for example to define that event X happens before event Y, or that event X occurs during event Y. Allen's Interval Algebra is used by a number of ontologies, including the DOLCE [77] upper ontology.

The DOLCE upper ontology is discussed in more detail in Chapter 9.

Semantic Web Rule Language (SWRL) Basic Temporal Built-ins support `xsd:date` and `xsd:dateTime` with Allen's Interval Algebra. The Advanced Temporal Built-ins uses the `temporal` ontology [64] to provide additional functionality, for example having different granularity levels.

The Dublin Core (DC) Terms ontology has a `temporal` property to describe temporal coverage of a resource with a range `periodOfTime`.

TopBraid Composer has a Calendar ontology that defines an `Event` and its `startTime` and `endTime` (as `xsd:dateTime`). This can then be used with the Calendar View widget in the editor.

In SPARQL Protocol and RDF Query Language (SPARQL), we can use the $<$ and $>$ operators on dates, for example

```
FILTER(?date > "2005-10-20T15:31:30"^^xsd:dateTime)
```

Using SPARQL Inferencing Notation (SPIN), can also cast a `xsd:dateTime` value to a string using the `fn:substring` function, for example

```
fn:substring(xsd:string(afn:now()),0,10)
```

8.2 INTERACTION EVENTS

We now turn our focus to how interaction events are modelled in the work described in this thesis. An interaction event happens at a specific time, is generated by a smart object and has an optional data value associated with the event.

An example of an event generated when an alarm is set is

```
:event-43495d51-29e3-11b2-807e-ac78eefc1f82
        rdf:type :AlarmSetEvent ;
        :generatedBy :phone1 ;
        :inXSDDateTime "2012-01-17T11:22:06.887+01:00"^^xsd:dateTime ;
        :dataValue "2012-01-17T12:00:00+01:00"^^xsd:dateTime .
```

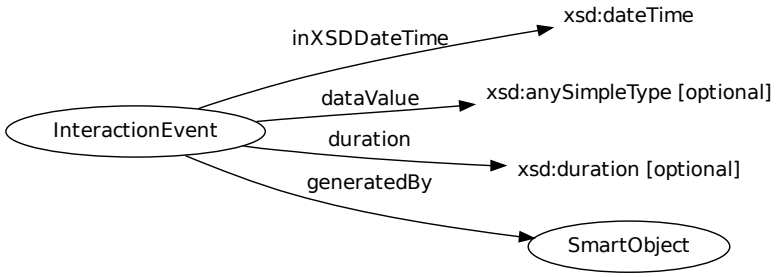

Figure 46: An interaction event as modelled in the ontology

DUL	EO	LODE	INTERACTION EVENTS
isObservableAt	time	atTime	inXSDDateTime
	place	inSpace	
hasLocation		atPlace	
hasParticipant	factor	involved	
involvesAgent	agent	involvedAgent	generatedBy
			dataValue

Table 7: Mappings between the various event models (adapted from [80])

A mapping between our interaction event model and the other event ontologies is shown in Table 7. Note that we do not model people or places in the current version of the ontology, as we consider these entities to be optional when describing interaction events.

The `duration` property is used to define the length of event. For example to increase the brightness of a lamp, we can generate an event to increase a value to a set maximum over a time period:

```
:event-43495d51-29e3-11b2-807e-ac78eefc1f83
        rdf:type :IncreaseLevelEvent ;
        :generatedBy ie:wakeup1 ;
        :inXSDDateTime "2012-01-17T11:23:06.887+01:00"^^xsd:dateTime ;
        :dataValue 255 ;
        :duration "PT3S"^^xsd:duration .
```

We also distinguish between *control* and *content*. Interacting with a device, e.g. pressing a "Play" button or moving a volume slider, is considered control and described using interaction events. Content, e.g. a song or a photo stored on the device, is referred to by where it exists on the device, as well as how it can be rendered using the media capabilities of the device.

We consider interaction events to be traceable, reversible and identifiable. Each interaction event has an associated timestamp and a unique ID that is generated when the event occurs.

Intentional, incidental and expected interactions were introduced in Section 2.5.1.

An intentional interaction, like pressing a light switch, is an interaction event if the light switch shares this information with other devices. Incidental or expected interactions, like the light turning on if the presence sensor is triggered, are also interaction events. System events, like a `TimeSetEvent`, which are invisible to the user are not considered to be interaction events.

8.3 CATEGORISING INTERACTION EVENTS

Also see the related work on task models in Section 2.4.

In the iStuff toolkit [6] *hierarchical event* structures were used to abstract low-level events into application-level events. We also introduce the notion of an event hierarchy, where low-level events are considered to be very *generic*, as they do not report a user's intention directly [59]. These low-level events first need to be transformed into intentional events—events that express user intention.

We build on the different interaction layers introduced in the related work of Section 2.3 to categorise interaction events. As an example, consider the case where a rocker switch, modelled as an interaction primitive on a mobile device, is used to control the volume of music in a room. One could start modelling the interaction on the physical level with a `ButtonEvent`, but it would be more meaningful to model it on the lexical level as a `ButtonUpEvent`. On the syntactic level this event could increment a quantity by one, while an event generated by a volume dial might have a discrete value attached to it. When this is combined with other device information, for example that the device is being used to stream music to the environment, we can infer on the semantic level that it is a `VolumeUpEvent`. When this is combined with other contextual information, for example that the device is currently connected to a speaker system in the same room, we can even infer on the task level that the music volume in the room should be set to a specific value with a `MusicVolumeUpEvent`, to which all connected devices can respond. We acknowledge that in most cases it may not be possible to make inferences on the goal level, which in this example could be the user's intent to set the music volume in the room to a level that is loud enough for everyone in the room to dance to.

Figure 47 shows how the interaction model levels relate to the concepts in the ontology that was developed through the various iterations. A `ButtonEvent` describes a user interaction on the alphabetical level, but carries very little meaning. For example, was the button switched up or down, or was it pressed? If we know that the button was switched up, we have a lexical token that describes the interaction, but it still needs to be combined with other contextual informa-

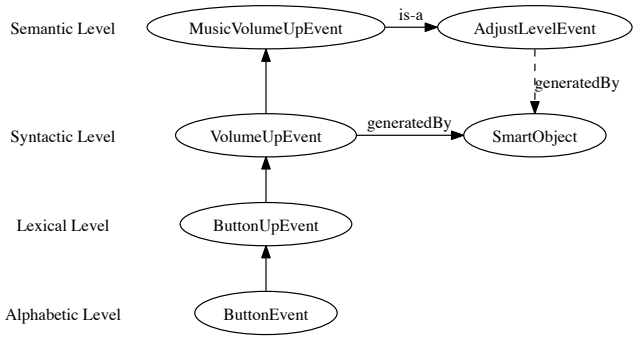

Figure 47: How the interaction model levels relate to the ontology

tion to determine the user's intention. On the other hand, if the interaction is described as a `VolumeEvent`, the user's intention is described on a syntactic level, and it becomes possible to map the event to a set of predefined semantic events, for example an `AdjustLevelEvent`. Using semantic reasoning, we can also infer that this `AdjustLevelEvent` was generated by the same smart object.

8.3.1 *System events*

When a smart object first subscribes to the smart space, it specifically listens for events that are generated by other smart objects connected to it. This means that we also need some way of distributing system-wide events that all devices listen for. As an example, consider the `TimeSetEvent`. When the user sets the time on one device, we want the time to be immediately updated on all the other smart objects in the smart space, even if they are not connected to the device that generated the `TimeSetEvent`. If we define `TimeSetEvents` as a subset of `SystemEvents`, each smart object only need to subscribe to events of type `SystemEvent`.

8.3.2 *Feedback*

When setting an alarm for example, augmented feedback should be provided on all devices. Functional feedback, i.e. the alarm sound when an alarm fires is triggered, is delayed. This means that augment functional feedforward should be provided. We thus define two types of feedback events:

Preview events and indicator events were first introduced in Section 5.4.1.1.

- `PreviewEvent` - generated when a possible connection is being explored, displaying the possible functionalities enabled by the connection, i.e. augmented functional feedforward.

- `IndicatorEvent` - augmented feedback when smart object is connected and there is no immediate functional feedback, e.g. a sink "beeping" when the alarm is set on the source; used to confirm actions.

The type of feedback required depends on the functionality of the connection. It is important for the feedback to coincide in time and modality with the event generated, as to maintain the causal link that is perceived by the user.

The device used to make the connection, for example the Connector object, creates a temporary connection to the devices to be connected in order to generate a `PreviewEvent`. This `tempConnectedTo` property is a sub-property of the `connectedTo` property. This means that the smart objects will handle it as if it is a regular connection, and when the Connector object removes the `tempConnectedTo` relationship, the inferred `connectedTo` relationship will disappear as well.

8.3.3 *Discussion & Conclusion*

Tungare et al. [84] defined a *task disconnect* as "the break in continuity that occurs due to the extra actions outside the task at hand that are necessary when a user attempts to accomplish a task using more than one device." Kuniavsky [44] states that consistency is the key aspect in creating task continuity across devices, and that *interaction vocabularies* have recently emerged as a way of consistently interacting with a range of devices. This ranges from simple vocabularies of light patterns and motion as used by the (now discontinued) Nabaztag Internet-connected rabbit that could compose "sentences" with more complex meaning, to a set of visual icons by Timo Arnall [5] that represents various kinds of touch-based RFID interactions. Arnall's touch-based vocabulary is shown in Figure 48. Arnall categorises his vocabulary of visual icons into four classes of interactions:

- Circles, of which the "Aura" is an example
- Card, of which the "Card in hand" is an example
- Wireless, of which the "Wireless dot" is an example
- Mobile, of which the "Phone beaming" is an example

The "Aura" icon communicates both the near-field communication capabilities of the technology, but also indicates that the physical object has capabilities beyond its form. The icons with cards or mobiles could improve consistency for a wide range of users which use these technologies on a daily basis.

These interaction vocabularies try to smooth over task disconnects through consistency. We argue that by having a vocabulary, or ontology, for interaction events could improve consistency in ubiquitous

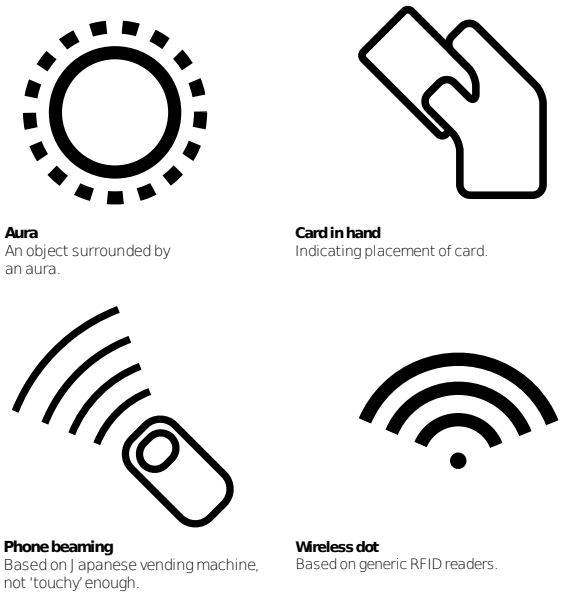

Figure 48: Examples from Arnall's "A graphic language for touch" (adapted from [5])

computing environments. This is the intention of the work in this chapter — to provide an ontology of interaction events that improves consistency for users, as well interoperability between devices.

Part IV

APPENDIX