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.

5 DESIGN ITERATION III

Making everything visible is great when you only have twenty things. When you have twenty thousand, it only adds to the confusion.

— Don Norman [77]

The goal of the final iteration was to extend the scenarios developed in the previous iterations to a new domain, while still making use of the smart objects and concepts that have been developed thus far. This would allow for testing the general applicability of the concepts and techniques, while still being able to reuse some of the devices we have already developed.

5.1 REQUIREMENTS

The use case scenario in this iteration revolves around a person's evening routine before falling asleep. It is a cross-domain scenario that extends the media domain into the sleep domain, and enables the exchange of different types of information. The domain of sleep was chosen for several reasons:

- Sleep is important for physical and mental well-being — an important application area of our research group at TU/e.
- The sleep domain is targeted by a number of recent Internet of Things (IoT) devices that record and share data and can be accessed through their Application Programming Interfaces (APIs).
- The sleep domain allows us to reuse some of our existing work on media sharing and lighting, extending it into a new domain.

In the fitness and sleep domains there are a plethora of devices that are well-known to the IoT community but that are not interoperable, such as the Withings WiFi body scale[1], Fitbit activity tracker[2], Nike FuelBand fitness monitor[3] and the Zeo sleep monitor[4].

We would like to be able to not only visualise the data coming from these devices, but also to change the behaviour of other devices that are connected to these devices. For example, we could use the data coming from a sleep monitor to to change the behaviour of a light in the room, or the alarm on a mobile phone. We distinguish between

1 http://www.withings.com/en/bodyscale
2 http://www.fitbit.com/
3 http://www.nike.com/fuelband/
4 http://www.myzeo.com

a number of subdomains within the area of well-being, as shown in Figure 22.

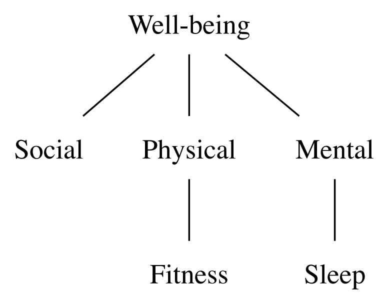

Figure 22: Sub-domains of well-being

The use case in Iteration III consists of several devices. It includes:

- an Android smart phone – Samsung Nexus S;
- an internet radio – Logitech Squeezebox Radio;
- the lamp from Section 3.3.2;
- a sleep monitor – Zeo Sleep Manager; and
- an Android tablet – Samsung Galaxy Tab 10.1 WiFi.

We purposefully did not define a narrative for this use case, to refrain from only implementing the functionality described in the narrative. Instead, we looked at the meaningful ensembles we could create with the devices, attempting to allow for *emergent functionalities* to surface by sharing device capabilities and interaction events.

The use case was implemented in the master bedroom of the Context Lab of TU/e, a lab with a setting that resembles a real home. Implementing the use case in-context allowed us to see its behaviour and implications in a realistic setting, giving insights that are regarded more valuable than obtained when building a setup on for example one's office desk.

5.2 ONTOLOGY DESIGN

In this iteration the earlier ontologies were consolidated into a single ontology. This helps make the ontology more manageable and removes the "cruft" of legacy statements that build up over time.

The first design decision of this iteration was to introduce the notion of directionality. This gives additional meaning to the devices, which now need to be modelled as *sources*, *sinks* or *bridges*. A music player is an example of a smart object that acts as a source when connected to a speaker, which in turn acts as a sink. When transitivity is introduced, a smart object can act as both a source and sink, which we define as a bridge. For example, consider the case where the speaker is connected to another speaker, which then also plays back the same music. The first speaker then acts as a bridge.

We can infer that a smart object is a sink using

$$\text{Sink} \equiv \text{SmartObject} \sqcap (\text{functionalitySink} \; \exists \; \text{Functionality})$$

where the symbol ∃ is used to denote the existential restriction that `functionalitySink` is some kind of `Functionality`. A bridge is inferred using

See Table 8 on page 130 for more details on the symbols and syntaxes used in this thesis.

$$\text{Bridge} \equiv \text{Sink} \sqcap \text{Source}$$

A semantic transformer is a virtual component that is not physically addressable and is therefore not considered to be a smart object. However, it is a bridge, as it acts as both a source and a sink. A smart object is a physical object first, with a digital representation added later.

Semantic transformers were first introduced in Section 4.2 and are discussed in more detail in Section 6.4.

Other areas where the ontology was improved include the modelling of device capabilities (Chapter 7) and the modelling of events (Chapter 8). These improvements are discussed in more detail in the relevant chapters.

5.3 DEVICE DESIGN

In this iteration, we reused both the ambient lighting system from Section 3.3.2, as well as the Connector object from Section 4.3.2. For the Squeezebox radio and Android devices new Knowledge Processor (KP) software was developed.

5.3.1 *Squeezebox*

The Squeezebox radio, shown in Figure 23, can be controlled via a Telnet interface over WiFi.[5]. For example, the accepted parameters for setting an alarm are shown in Table 5.

On startup, the Squeezebox KP connects to the smart space, registers the capabilities of the device, checks for existing connections and listens for new connections. It also subscribes to new system events. It then connects to the Squeezebox device via the Telnet-over-WiFi in-

System events are discussed in more detail in Chapter 9.

5 On Squeezebox Server, the interface documentation is available from `Help` ⇒ `Technical Information` ⇒ `The Squeezebox Server Command Line Interface`

Figure 23: Logitech Squeezebox Radio

PARAMETER	DESCRIPTION
dow	Day of week (0 – 6, starts on Sunday)
time	Time since midnight in seconds
repeat	1 or 0
volume	0 – 100
url	Squeezebox Server URL of alarm playlist
id	The ID of an existing alarm (optional for new alarms)

Table 5: Accepted parameters for Squeezebox `alarm` Telnet command

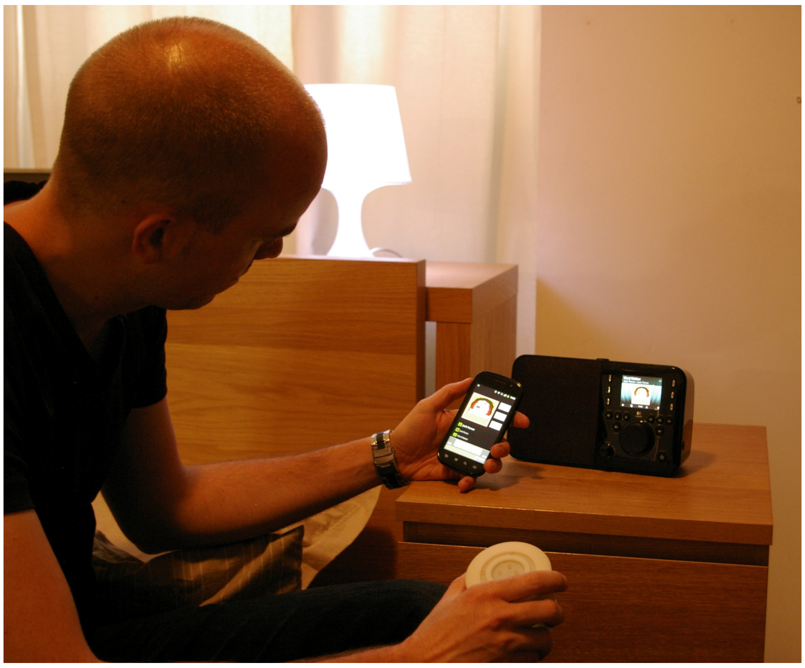

Figure 24: Playing music from the phone on the Squeezebox radio

terface, subscribes to new events generated by the device and enters an event loop.

Media events were introduced in Section 3.4.2.

When a new alarm is set on the device, the KP converts the date and time to XML Schema Definition (XSD) format and generates a new `AlarmSetEvent`. When an alarm is triggered, an `AlarmAlertEvent` is generated. If the alarm is dismissed on the device, an `AlarmEndEvent` is generated. When an alarm is deleted, an `AlarmRemoveEvent` is generated.

When an `AlarmSetEvent`, `AlarmRemoveEvent`, `AlarmEndEvent` is received from another device, the corresponding action is performed on the device. The device also responds to media events like `PlayEvent`, `PauseEvent` and `PlayEvent`.

5.3.2 *Android mobile devices*

The KP developed for the Android devices was tested on both the Google Nexus S phone and the Samsung Galaxy Tab.

To improve software reuse and not reinvent the wheel, we wanted to make use of the stock applications on the phone, like the Clock app and the Music app (shown in Figure 24), instead of developing our own. On Android, it is possible to run a service as a background process that listens for events generated by other applications. A *broadcast receiver* listens for *broadcast intents*, which are public intents broadcast from activities to registered receivers. A receiver registers for a broadcast intent by listing it in its intent filter in the manifest file. Broadcast intents sent by Android applications can be received by all other applications, which is done by creating a broadcast receiver.

Android activities run inside applications.

When the alarm is triggered in the alarm app on the mobile phone,

The alarm app on the Google Nexus S phone is called `DeskClock` and was developed by Google. There also exists a version for earlier Google phones called `AlarmClock`.

a

```
com.android.deskclock.ALARM_ALERT
```

broadcast intent is generated.

A broadcast receiver handles such an intent using

```
@Override
protected void handleBroadcastIntent(Intent broadcastIntent) {
        String action = broadcastIntent.getAction();
        if(action.equals( "android.intent.action.ALARM_ALERT" )) {
                addEvent( "AlarmAlertEvent" );
        }
}
```

Note that this intent is not supported by all Android devices, as different devices may have different default alarm applications. It did, however, work on both the Google Nexus S phones and Samsung Galaxy tablets that we tested. To determine when an alarm was changed, we made use of the

```
android.intent.action.ALARM_CHANGED
```

broadcast intents. It is also possible to read the next alarm that will triggered from the system settings, using

```
System.Settings.NEXT_ALARM_FORMATTED
```

To determine if a song is being played using the Android Music app, we used the

```
com.android.music.playstatechanged
```

broadcast intent.

We used a Lighted Greenroom [57] pattern to launch a long-running service from a broadcast receiver, without the operating system throwing an Application Not Responding (ANR) message. ANR specifies a 10-second response limit for a broadcast receiver, after which it is deemed unresponsive. By launching a separate service that handles generation of events based on broadcast intents, we have a workaround to this problem. This allows us to listen for broadcast intents from applications like the music player and the alarm clock.

5.3.3 *Wakeup experience service*

In the sleep use case, music can be shared between the smart phone and the internet radio. Alarms can be shared between the phone and the internet radio, the internet radio and the lamp as well as the phone and the lamp. Because the lamp has only `LightOn/LightOff` and `AdjustLevel` capabilities, the most basic functionality of the lamp responding to an `AlarmEvent`, would be to turn on at the time that the event occurs. However, a wakeup service can be connected that *transforms* an `AlarmSetEvent` into a wakeup experience, sending a sequence of `AdjustLevelEvents` to the lamp. This wakeup service then functions as a semantic transformer, transforming one type of value into another in a meaningful way. Semantic transformers are virtual entities and therefore they do not have a physical presence, in contrast to smart objects that must have a physical representation. Therefore, the use of a semantic transformer is automatically inferred based on its capabilities, as it cannot be physically connected to other devices by the user.

To create a wakeup service, an `AlarmSetEvent` would have to trigger an `AdjustLevelEvent` event with a `dataValue` that increases from 0 to 100 over a period of 30 minutes *before* the alarm sounds. Another requirement is that it should work with any light and any alarm in the smart environment.

This wakeup service has similar functionalities as a Wakeup Light (e.g. as sold by Philips[6]) which means it starts increasing its light level over a 30 minute time-period, reaching full intensity (as calibrated) at the set alarm time. The semantic connection between the phone's alarm and the dimmable light is an example of how such a connection can have emerging functionality, which does not exist without the connection and the wakeup service.

This opens up many possibilities for users, as they may connect other lights, and potentially even other devices such as a networked thermostat, to either the alarm or the dimmable lamp, creating their own wakeup experience. Whether such emerging functionalities are possible obviously depends on the way the smart objects are implemented. For example in our implementation, the dimmable lamp is described as a sink, which means it is only capable of accepting input. If it was described as a source as well, sharing for instance its on/off state or its current light value, it could act as a bridge and allow for more interesting configurations. Users may also connect the sleep monitor as an alarm source, helping them to wake up at the right time in their sleep cycle.

5.3.4 *Zeo*

The Zeo sleep monitor is shown on the left-hand side of Figure 25. The Zeo headband uses three silver conductive fabric sensors to collect Electroencephalography (EEG) signals while a person is sleeping. The signals are amplified and features are extracted using a Fast Fourier Transform (FFT). An Artificial Neural Network (ANN) is then used to estimates the probability of a person being in a certain phase

6 http://www.philips.co.uk/c/wake-up-light/38751/cat/

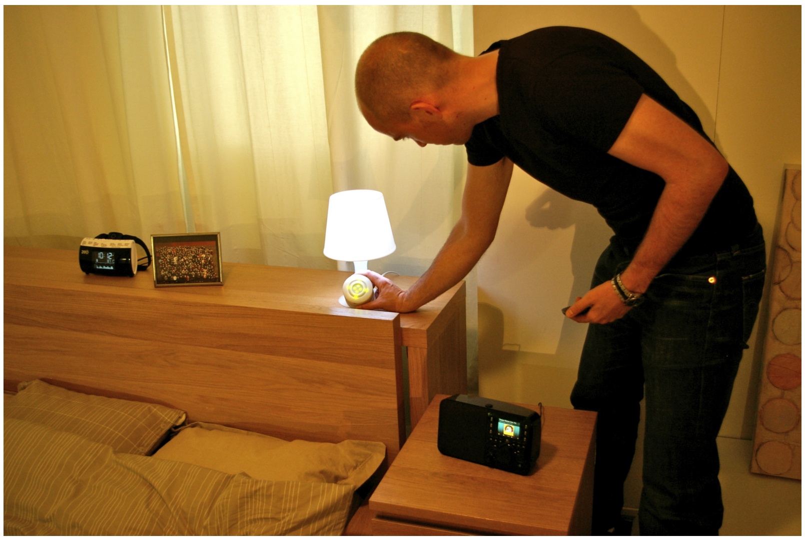

Figure 25: The sleep use case scenario, with the Zeo sleep monitor on the left, the dimmable light and the Connector object in the middle, and the Squeezebox on the right

of sleep[91]. The sleep stages are Awake, Rapid Eye Movement (REM) Sleep, Light Sleep, Deep Sleep or Undefined.

Sleep data is stored on an SD card on the device and can be uploaded to the Zeo MySleep[7] website. Zeo created the Data Decoder Card library[8] that allows developers to decode the sleep data without uploading the data to the MySleep website. We also built a USB cable that connects to a serial port on the back of the device. With this cable you can access the raw data coming from the headband sensor.

When the device is connected via a USB cable, we have real-time access to the generated events. Events that could be interesting to other smart objects in the environment include:

- `NightStart` - time when first "Awake" hypnogram occurs
- `SleepOnset`
- `HeadbandDocked` and `HeadbandUndocked`
- `AlarmOff`, `AlarmSnooze` and `AlarmPlay`
- `NightEnd`

The Zeo Raw Data Library[9] is a Python library to read raw data from the Zeo serial port, using the following commands:

```
from ZeoRawData.BaseLink import BaseLink
from ZeoRawData.Parser import Parser
```

7 http://mysleep.myzeo.com
8 http://developers.myzeo.com/data-decoder-library/
9 http://sourceforge.net/projects/zeorawdata/

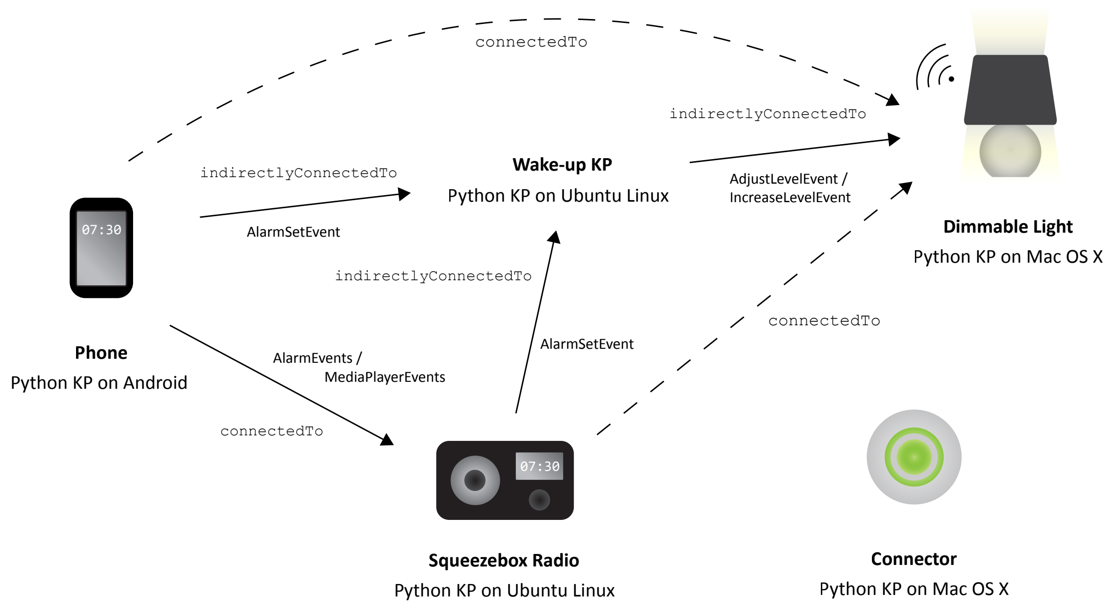

Figure 26: An overview of the sleep use case

```
# Initialize
link = BaseLink( '/dev/ttyUSB0' )
parser = Parser()

# Add callback functions
link.addCallback(parser.update)
parser.addEventCallback(eventCallback)
parser.addSliceCallback(sliceCallback)

# Start link
link.start()
```

5.4 IMPLEMENTATION

We started by implementing a very basic configuration, connecting the phone and the internet radio. Based on the capabilities of the devices, possible connections included sharing music player functionality and alarm clock functionality. After implementing the first basic functionalities, we gradually increased complexity by adding another smart object, the dimmable lamp, followed by the implementation of several types of interaction feedback.

In the sleep use case we did not make use of the Zeo and its KP implementation. It was viewed as a backup device which could be used to introduce additional complexity to the system if necessary.

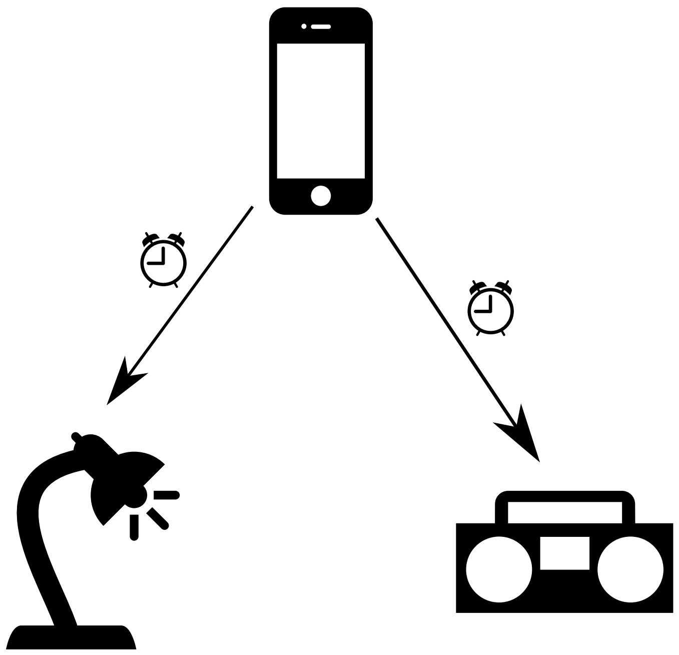

Figure 27: Alarm functionality of the phone shared with the radio and the lamp

5.4.1 *Feedback and Feedforward*

The different types of feedback and feedback as described in the Frogger framework was discussed in Section 2.5. We use feedforward to display a device's functional possibilities. We can use feedback to confirm user actions, using augmented feedback where direct functional feedback is not available.

In Van der Vlist's thesis [108] there is a similar discussion on feedback, that focuses on some of the more user-centred aspects of feedback.

When an alarm is set on the phone, augmented feedback should be given on all devices connected to the phone. For example, consider the setup in Figure 27, where the alarm is connected to both the lamp and the Squeezebox radio.

Immediate feedback only makes sense when the event and its feedback coincide in time and modality (e.g. audio, visual). When the generated event is a `SetEvent`, the event itself will occur sometime in the future, so we generate the functional feedforward as augmented feedback instead. For example, for an `AlarmSetEvent` we generate a `1s` alert sound on the Squeezebox radio as augmented feedback, providing functional feedforward of what will happen when the alarm is triggered. We also provide visual augmented feedback by displaying a popup message on the display for a few seconds. On the lamp feedback is given in the form of a short light pulse to confirm that it has been notified as well.

Feedback and feedforward need to be carefully designed when smart objects are interconnected. However, as the smart objects themselves are unaware of each other and, at development time, their designers did not know to what other devices users may connect the smart objects to, the total user experience cannot easily be designed. In this section we will describe how feedback and feedforward were used to enhance the user experience and enable devices that are in-fact unaware of one another, appear to show awareness of each other to their users.

Many solutions for interconnecting devices often employ the vendor lock-in *strategy, which enables manufacturers to have full control over their ecosystem of products and the resulting user experience.*

Augmented and functional feedforward

For semantic connections, functional feedback and feedforward can only be considered for the combination of source and sink. The source object has functional feedforward that may communicate its function. Only when both the source and sink object have been identified, is functional feedforward available for the semantic connection. Important to note is, that functional feedforward is derived from the intersection of functionalities of both the source and the sink. These functionalities could be ambiguous, as both source and sink may be multifunctional. If this is the case, users should make explicit what information or data they want to exchange by selecting the desired mode on the source object (e.g. selecting the alarm application on your smart phone to share the alarm time or go to a picture viewer when pictures should be exchanged), restricting the possibilities. If this is not possible, or a multifunctional smart object is connected when it is in idle mode, semantic reasoning could be used to match all meaningful capabilities of the source and sink objects.

Whenever users wish to make a connection, they have certain expectations. We can employ functional feedforward to influence these expectations. Additionally, we can enhance the user's understanding by explicitly adding augmented feedforward (i.e. augmented *functional* feedforward in contrast to augmented *inherent* feedforward). In the sleep use-case we employed augmented feedforward in the process of exploring connection possibilities i.e. before the connection is made. We do this by giving a *functional preview* on the sink object, viewing the functionality of the connection that is currently explored. Our reasoning is, that only when both source and sink are identified, we can speak of a semantic *connection* and, by giving the feedforward at the sink, we ensure that the sink object is in fact capable of producing this feedforward (i.e. has the necessary capabilities). Additionally the location of the feedforward corresponds to the location where the action (identifying the sink object) was performed. To do so, a `PreviewEvent` is generated when a possible connection is being explored, displaying the possible functionalities enabled by the connection.

Example 1. *When a user, after having identified the phone as a source object, identifies the internet radio as a sink, the display of the internet radio displays a message: "Alarm can be shared" and "Music can be played". Previews can also be less explicit, like briefly sounding an alarm and playing* *a short music clip. Note that the preview can be ignored or bypassed by establishing a connection.*

Example 2. *For exploring a connection between the internet radio and the dimmable lamp, the lamp simulates a wakeup sequence, increasing the light level from zero to its maximum intensity in a given period of time (in our implementation three seconds). This may be enhanced with simulating an alarm at the Squeezebox radio when the maximum of the intensity is reached.*

Practically, this means that the designer/developer of a smart object should design the response to a `PreviewEvent`. Technically, this is implemented by having the Connector object create 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 (which denotes a regular semantic connection). 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. The type of functionality the preview is for, is added to the preview event as a data value.

The system behaves differently depending on the type of relation between the smart objects. When there is an indirect connection, i.e. going through a semantic transformer, the preview event is sent to the semantic transformer (Figure 29) instead of the sink object directly (Figure 28). Additionally, a temporary connection is made between the semantic transformer and the sink, ensuring that the sink displays the correct feedforward when the `PreviewEvent` is received.

Functional feedback

In many cases functional feedback of a semantic connection is trivial, for example hearing sound from a speaker that was just connected to a media player, or seeing photos on a TV when it is connected to a smart phone. However, functional feedback may only be available at another place or at another time. If we for instance take the example of synchronising a phone's alarm with the alarm radio, the real functional result may be hearing the radio play a song at the alarm time that was set on the phone.

In such cases, the interaction designers should use augmented feedback as an *indicator* that the alarm time was successfully set. When a semantic connection exists between a source and a sink, actions at the source should also be indicated at the sink.

If the source and sink objects are in different locations, interaction designers should make sure that feedback is visible for a prolonged time period, or until it is dismissed by the user. This is to ensure that the indicator of the performed action will be noticed by the user. For the same reason, the order of connecting two spatially separated

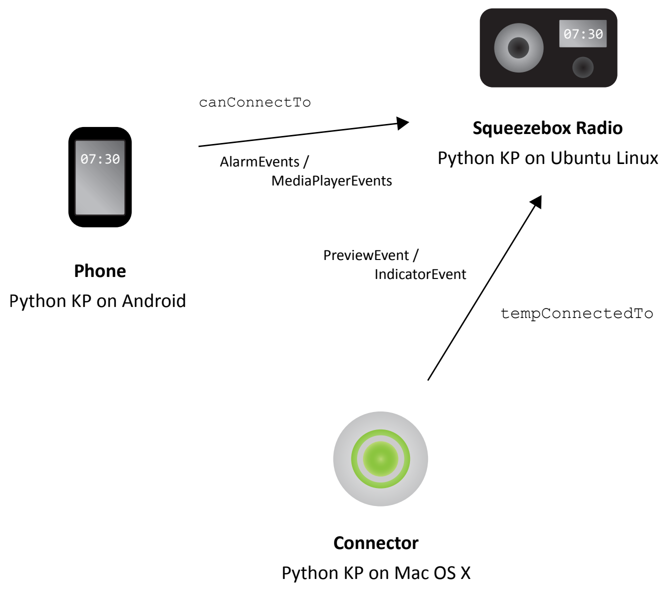

Figure 28: Temporary connections for a `PreviewEvent` when source and sink are directly connected

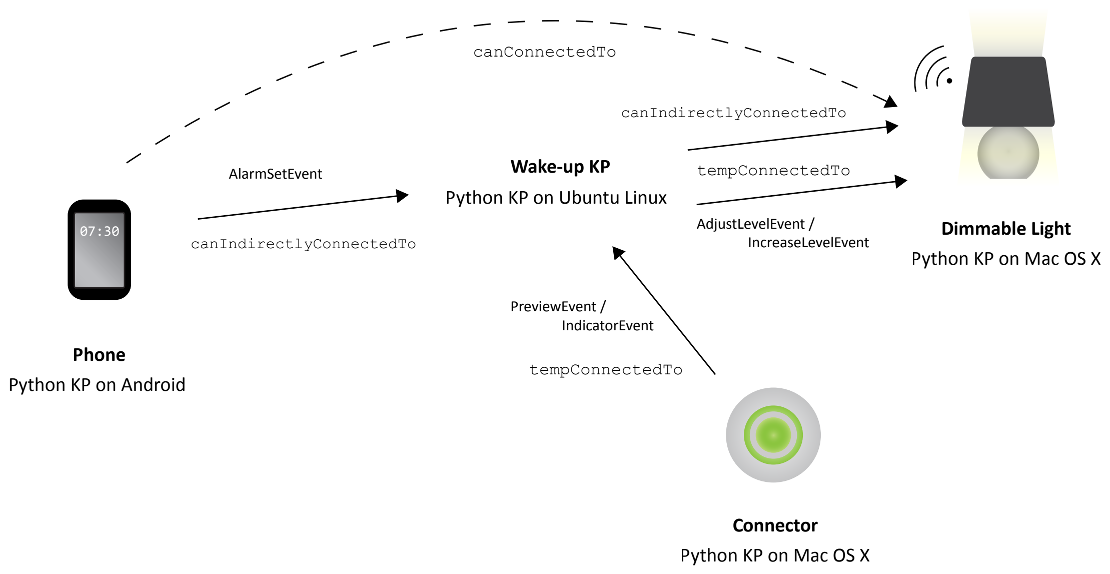

Figure 29: Temporary connections for a `PreviewEvent` when source and sink are connected via a semantic transformer

objects together is important, ensuring that establishing the connection happens in proximity of the sink, so that the feedback can be observed.

Example 3. *When music is playing on the phone and a connection is made between the phone and the internet radio, functional feedback is immediately given: the internet radio starts playing the same music, and an image of the album cover is displayed. The music on the source (phone) is muted, as the music playing on the internet radio is of a higher fidelity, and both share the same physical space. Context information, such as place/location, can be used to infer the correct behaviour.*

Augmented feedback

If there is no immediate link between action and function (e.g. functional result is delayed, information is given about an internal state change), augmented feedback can be used to provide this information. We use an `IndicatorEvent` to provide 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. 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.

When a connections exists and an action performed on the source that has no immediate functional feedback, augmented feedback is provided to serve as an *indicator*. This feedback is provided by the smart objects that are connected, in the modality that is supported by their interaction capabilities. Designers should aim for maintaining the modality of the augmented information across the smart objects. Additionally, ensuring that the feedback occurring at distributed objects coincide in time may strengthen the perceived causality of the link. Indicator events may also be used to indicate existing connections, e.g. when a user wishes to see what smart objects are currently connected to a source.

Example 4. *When the phone is connected to the internet radio and the internet radio is connected to the dimmable lamp, both the internet radio and the lamp gives augmented feedback when an alarm is set. The internet radio displays the alarm-set screen, confirming the alarm time and the dimmable lamp slowly flashes, to indicate that they are both connected and that the action on the source is confirmed.*

5.5 DISCUSSION & CONCLUSION

5.5.1 *Mismatches between device states*

Interaction events (Chapter 8) cause device state changes. When smart objects are interconnected, mismatches in device states may occur, as not all interaction events cause the same state transitions in all smart objects. The design decision to describe the interactions in terms of events as opposed to states was based on the idea that states can be logically inferred from events. Exchanging the events still leaves some autonomy to the smart object (or its developer) to decide what to do with the event. However, only sharing events is not always enough to create consistent behaviour.

It is the responsibility of the source to communicate all state changes in the form of events, in order for the sink to keep in sync. Even then, it is still possible for two smart objects to be in different states while connected. Consider the case where the mobile phone is connected to the internet radio, sharing both alarm and music functionality. The user opens the music player on the mobile phone and presses Play, which causes the music to play back on the internet radio. As shown in Figure 30, an `AlarmAlertEvent` generated by the mobile phone will cause the internet radio to go into an `Alarm` state. When the alarm is dismissed on the mobile phone, the internet radio will go into a `Pause` state, as this is the default behaviour when the alarm is dismissed on the internet radio itself. The mobile phone, however, is now in the `Play` state. To prevent this state mismatch, the mobile phone should not only send the `AlarmDismissEvent` that was generated through the user interaction, but also a `PlayEvent` to indicate that it is now playing music again.

5.5.2 *Setting time on devices*

For devices to interoperate with one another in a meaningful way, it is imperative that they have the same concept of time and that their clocks are synchronised. For something that should be very trivial, setting the time on devices is surprisingly difficult. Consider the devices we used in our last scenario:

- The Zeo sleep manager does not allow for the time to be set remotely - it can only be set manually on the device itself.
- The Squeezebox radio retrieves the time from the server that it is connected to - you therefore need to be able to set the time on the computer running the server.
- On Android devices there are documented methods to set the time, e.g. `setCurrentMillis()`, but due to security restrictions

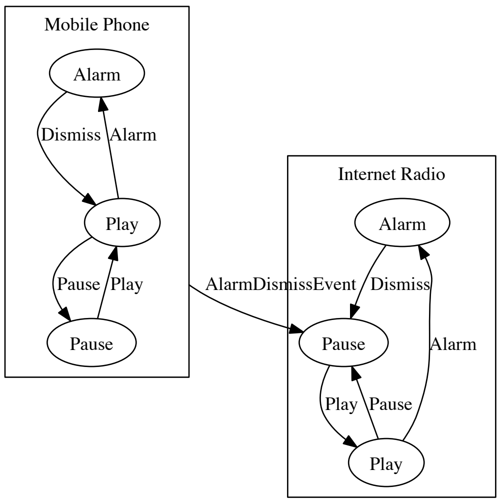

Figure 30: State mismatch between phone and internet radio

non-system applications are not allowed to access these methods.

`TimeSetEvent` is a type of system event. For more information on system events, see Section 8.3.1.

Our workaround was to set the time on the Android phone, which will send the time value with a `TimeSetEvent` to the `SIB`. When a new `TimeSetEvent` is received, the Squeezebox KP updates the system time of the Squeezebox Server, requiring super user permissions. The Squeezebox is then restarted to synchronise the local time with that of the server.

5.5.3 *Conclusion*

The sleep use case acted as an evaluation of the completeness and applicability of the concepts and techniques described thus far. These approaches were distilled into a theory of semantic connections, which forms the basis of the next chapter. The implemented use case evaluated:

- whether the defined concepts and techniques are sufficiently defined to use them to implement the required functionalities;
- whether the defined concepts and techniques can be used universally (for different use cases); and
- whether the defined concepts and techniques form a complete set to describe the behaviour of semantic connections.

Additionally, the implemented use case can serve as an example of how the theory described in the next chapter is used in a relevant and contemporary setting.

We consider the length of the verbal descriptions in Section 5.4 above to become unwieldy when describing more complex situations. There is a clear need for some kind of diagram notation to describe these situations. In the next chapter we introduce a theory of semantic connections that was created to help solve this problem. We first describe the concepts that are central to this theory, and then introduce a diagram notation based on Finite State Machines (FSMs), that can be used to model and explain the different concepts and situations.

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.

Part IV

APPENDIX