Intelligibility Required: How to Make Us Look Smart Again

Jo Vermeulen, Kris Luyten and Karin Coninx
Hasselt University – tUL – iMinds
Wetenschapspark 2,
B-3590 Diepenbeek, Belgium
[jo.vermeulen, kris.luyten, karin.coninx]@uhasselt.be

ABSTRACT

Users often become frustrated when they are unable to understand and control a ubicomp environment. Previous work has suggested that ubicomp systems should be *intelligible* to allow users to understand how the system works and *controllable* to let users intervene when the system makes a mistake. In this paper, we identify several design considerations for supporting intelligibility and control in ubicomp environments. We show these considerations are also applicable and necessary beyond ubicomp. We position examples of existing solutions in the design space that is obtained from combining these dimensions and show how it can be used to explore design alternatives for supporting intelligibility and control.

Author Keywords
intelligibility; control; end-user configuration; ubicomp; context; feedforward.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Human Factors; Design.

INTRODUCTION

Ubiquitous computing (ubicomp) systems are generally *context-aware*, which means they act based on *context* [9]: implicit input collected from the environment [21]. These systems thus often act without explicitly involving the user, which may leave users surprised as to why the system behaves in a certain way. Moreover, system actions are usually a result of complex reasoning about context data, which might be hard for users to grasp [12].

However, being difficult to understand is only part of the problem. Context-aware systems have been shown not be infallible. They are bound to sometimes make mistakes because of the inevitable incompleteness of context information [4,6]. It is therefore important that users are able to correct the system if it makes a mistake. Failing to do so will eventually result in users who feel out of control, and might cause them to lose trust in the system [2].

Bellotti and Edwards argue that the more we try to get systems to act on our behalf, especially in relation to other people, the more we have to watch every move they make [4]. They have proposed a number of design principles to tackle these problems, including *intelligibility* (what others have called *scrutability* [5]) and *control*. They argue that context-aware systems should be *intelligible* by informing users about the system's understanding of the world and should offer users *control* in order to recover from possible mistakes.

In his book *The Psychology of Everyday Things* [18], Donald Norman introduced the Stages of Action model. The design principles that come into play to effectively bridge the Gulfs of Execution and Evaluation are widely recognized and adopted for traditional software, but are not always sufficient for ubicomp systems. First, *visibility* falls short since a lot of computing is hidden in the environment of the user and sensors that are hardly noticeable are used to perform part of the interaction. Moreover, ensuring the user can form a *good conceptual model* of the system is cumbersome for ubicomp systems given the complexity these systems tend to exhibit. These complexities are often hidden for the users, but unmistakeably present in the software architecture, which is often distributed, embedded in the environment and designed for simultaneous usage. Finally, besides informative *feedback* that tells users what has happened, ubicomp systems might also need to convey to users what is going to happen in the future. Intelligibility helps to overcome the difficulties of interacting with these systems, by revealing how the software acts and reacts.

In this paper, we explore how we can help users to understand how ubicomp systems work, and how we can support them in configuring and correcting the system's behaviour.

BACKGROUND

Bellotti and Edwards [4] state that *intelligible* context-aware systems are able to represent to their users what they know, how they know it, and what they are doing about it. Dourish proposed the idea of reflective *self-representations* that are generated by a system and reliably describe its state, while also allowing users to affect that state and *control* the system's behavior [11].

There are many approaches to providing intelligibility and control, some quite subtle. For example, Google Maps uses a growing or shrinking "blue circle" to convey how confident it is of the user's current location. Other basic examples of intelligibility can be found in recommender systems. Services like Amazon, the App Store, or Youtube provide recommendations of related content to their users. Users can also ask the system why a certain item (e.g., a book, app or video) was recommended to them. Additionally, the system offers control to users: they can affect the recommender engine's behaviour by indicating that they are not interested in those recommendations.

Several systems have been developed to support end-users in controlling, configuring or programming their ubicomp environments (*e.g.*, [8,20]). Further investigation is necessary, however, to explore which interaction techniques and user interfaces are best suited for this purpose. There are a number of ubicomp systems and architectures that extend intelligibility and control to end-users. Cheverst's IOS system [5] shows confidence levels and visualizations of decision tree rules and allows end-users to manipulate system parameters. Situations [10] automatically supports simple intelligibility and control user interfaces and also allows designers to create application-specific user interfaces.

Lim, Dey and Avrahami [15] investigated if why (not) questions could be used to improve the intelligibility of context-aware systems. Their results suggest that allowing users to pose why (not) questions about the behaviour of a context-aware system would result in better understanding and stronger feelings of trust. In a later study, Lim and Dey [14] investigated the different information needs users have for context-aware applications under various situations, recommending amongst others that why questions should be made available for all context-aware applications.

Ju et al. [13] describe a design framework for reasoning about transitions between implicit and explicit interaction. They discuss three interaction techniques that allow users to overcome errors in system's proactive behaviour: *user reflection*, *system demonstration*, and *override*. The first two can be seen as interaction techniques for improving intelligibility, while the latter is a technique for providing control.

Coutaz [7] proposed the *meta-User Interface* (meta-UI) concept, which is essentially a user interface to support intelligibility and control in smart spaces. Coutaz analysed several existing systems and argues that there should be more attention towards control by end-users and to embedding meta-UIs within domain-specific applications.

DESIGN SPACE

The systems that were discussed in the previous section only represent a single point in the larger design space of possible techniques to provide intelligibility and control. In order to get a better idea of the different possibilities and the design choices that play a role when developing interfaces for intelligibility and control, we introduce a design space consisting of six dimensions, as shown in Figure 1:

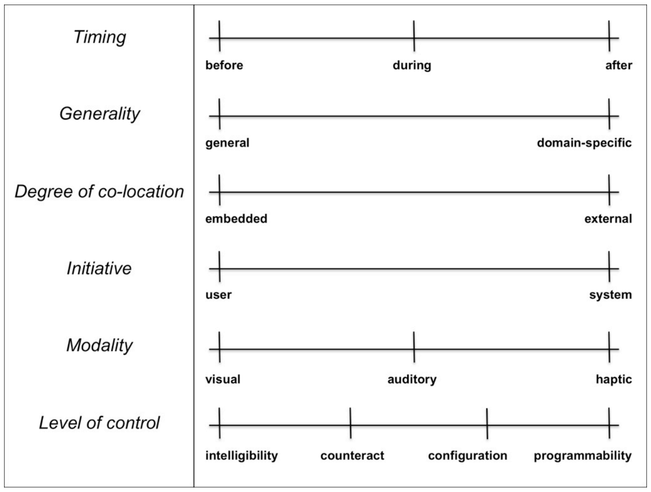


Figure 1: Design space for intelligibility and control.

We begin with a brief overview of each of these six dimensions, after which we discuss them in more detail and provide examples.

Timing: Intelligibility and control can be supported at different times during the interaction: *before*, *during* and *after* events take place.

Generality: User interfaces and interaction techniques for intelligibility and control can be *general* or *domain-specific* (e.g., techniques for visualising location errors in navigation systems).

Degree of co-location: Support for intelligibility or control might be *embedded* or integrated with the rest of the user interface versus *external*, when users are required to switch to a separate interface.

Initiative: Users may need to explicitly request intelligibility information or invoke control techniques manually (*user*), or might automatically be presented with these features when necessary (*system*).

Modality: Several modalities can be used to help users to understand or control the system (e.g. *visual*, *auditory*, *haptic*).

Level of control: The level of control end-users can exert over the system varies from *intelligibility*, where no additional control is added beyond intelligibility, over *counteract*, where users can perform the opposite action (*e.g.*, undo), to *configuration,* where users can tweak predefined system parameters, and *programmability* where users can themselves (re-)define how the system works.

Timing

Intelligibility information can be provided at different phases during the interaction with a ubicomp system. For example, consider the case where the system would perform a certain action automatically given a certain trigger (e.g., showing a personalized calendar on a proximity-aware display when a user approaches it). There are different points in time at which intelligibility information can be provided:

- *Before* the action: Users could be offered information before the action would take place, allowing them to anticipate the system's behaviour.
- *During* the action: The system could visualize events and actions when they happen, to allow users to better understand the flow between different components in the system (e.g., how different sensors work together).
- *After* the action: The user could be offered intelligibility information after the action has been performed, for example to explain why the system took that action.

For example, Ju et al.'s proximity-aware Range whiteboard [13] provides intelligibility and control *before* and *during* system actions. Range uses a *system demonstration* technique where the system shows the user what it is doing, or what it is going to do. When switching between ambient mode and the drawing mode, Range uses a transition of the whiteboard's contents to call the user's attention to the mode change, instead of suddenly switching. Moreover, while the whiteboard is transitioning between modes, users can grab the moving contents to cancel the mode switch (*override*). Additionally, Lim and Dey's concept of *what-if questions* [16] is intelligibility information that is provided before the action.

An example of intelligibility information that is provided after the action, are *why questions*. Ko and Myers developed the Crystal application framework [17] that allows programmers to support why questions in their applications. A word processor could, for example, allow users to pose questions about its more complex formatting behaviour (e.g., "Why is this text bold?"). Why questions have also been used for context-aware systems [16] and ubicomp environments [26].

Generality

Intelligibility or control techniques can be general or domain-specific. A simple example of a domain-specific intelligibility interface is the way location-based services present the user's current location together with the level of uncertainty [1]. For example, the blue circle in Google Maps gives users an indication of how certain the system is of the user's current location, depending on the size of the circle. This interface tells users that it knows they are located somewhere in the circle, but it does not know precisely where they are located.

While domain-specific interfaces might limit flexibility and reuse, they might be easier for users to understand as they provide a better expressive match. It is, for example, easier to estimate the location error using a circle on a map than to try to interpret an error percentage. Domain-specific interfaces can be more easily integrated into a specific application (see also: *co-location*), which might help users remain in the flow of their current task.

Other examples of domain-specific intelligibility interfaces are gesture guides, such as OctoPocus [3]. OctoPocus helps users perform gestures by continuously showing the possible remaining gesture paths. Similarly, Ju Lee and Klemmer's implementation of *user reflection*, *system demonstration* and *override* for the Range whiteboard is specifically designed for proximity-aware whiteboards.

An example of a generic interface for intelligibility and control is PervasiveCrystal [26] (see: next section), which provides users with the possibility to pose why and why not questions about any event occurring in a ubicomp environment and offers simple control primitives.

Co-location

The co-location dimension refers to the level of integration between an interface for intelligibility and control, and the application in which it is used. Intelligibility or control could be offered in a separate interface (*external*), or could be an integrated part of the application (*embedded*). In her discusson about meta-user interfaces, Coutaz [7] calls this dimension the "level of integration".

OctoPocus [3], Ju et al.'s techniques [13], and the Google Maps location error visualisation are all examples of *embedded* intelligibility interfaces. *External* interfaces tend to offer more possibilities and flexibility, but, unlike embedded interfaces, require the user to interrupt their task and switch to a separate interface. External interfaces tend to be useful for controlling or understanding high-level, generic components of a system. An example of an external interface is Dey et al.'s a CAPpella tool [8] that allows users to program a context-aware system by demonstrating its desired behaviour.

Initiative

The initiative for showing information to improve the users' understanding can be taken by the system itself or this information can be available upon request by the user. When the system takes the initiative, it could reveal information to draw the user's attention to a certain event, as with Ju et al.'s *system demonstration* technique [13]. Alternatively, the system could provide users with an option to receive detailed information if they need it, similar to the way services like Amazon can explain why a certain products were recommended to the user.

There might be several arguments for choosing between these two strategies. Automatically providing information all the time might be distracting or even annoying for the user, depending on the amount of detail that is provided. Still, it can be useful in select cases to have access to very detailed information to *debug* the system's behaviour and understand deeper details of how the system works. In this case, we would probably like to leave the initiative of showing this information to the user, so that the information is only there when necessary. On the other hand, simple and informative feedback that explains to users what the system is doing might be useful to show at all times, even for expert users.

An elegant way of supporting both novices and experts, and thereby combining system and user initiative, can be found in systems like OctoPocus [3]. This kind of system waits for a certain time before it presents the intelligibility interface, so that experts who already understand how the system works can perform actions very efficiently, but can always slow down when they are unsure.

Modality
Depending on the domain and the context of use, different modalities might be preferred (*visual, haptic, auditory*). For example, when the users need their visual attention elsewhere (e.g., while driving), intelligibility or control might be better provided using another modality. Most systems typically only support intelligibility or control using the *visual* modality.

Level of control
There are increasing levels of control that end-users can exert over the system. The most basic level of control is only *intelligibility*, where no additional control functionality is provided beyond intelligibility. Note that this still allows users to intervene by changing their own behaviour based on their understanding of how the system works. For example, because we understand how a motion-controlled light works, we know that we can wave our hands to turn the light on again when it goes out. An example of this level of control would be the availability of only a *why questions* interface. Based on the understanding gained by posing why questions, users could then alter their behaviour to exert control.

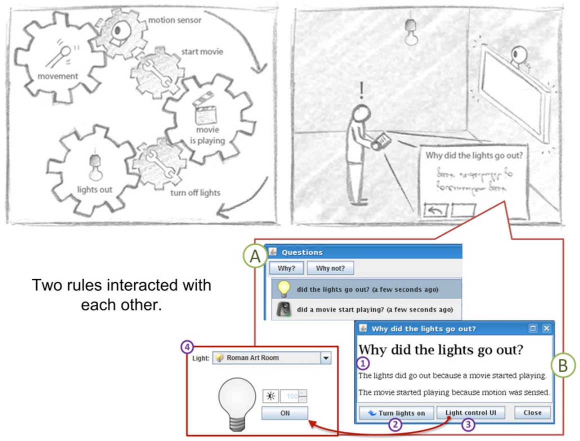
The next level of control is *counteracting*. Systems that provide this level of control only allow users to revert the system's actions (e.g., *undo*). An example of such as system is PervasiveCrystal [26] (see: next section). Next, systems that allow users to tweak predefined system parameters feature the *configuration* level of control. An example of this kind of system is Dey and Newberger's Situations framework [10].

The most advanced level of control, *programmability*, is available when users can themselves (re-)define how the system works, such as in Dey et al.'s a CAPpella [8].

INTELLIGIBILITY APPS: SOME SAMPLES FROM THE DESIGN SPACE

PervasiveCrystal
PervasiveCrystal [26] is a system that allows users to understand the behaviour of a ubicomp environment by posing *why* and *why not* questions. PervasiveCrystal can reason about the causes and consequences of system and user actions, based on a rule-based behaviour model, and uses this information to automatically generate a list of why and why not questions. PervasiveCrystal is built on top of ReWiRe [22], an existing framework to dynamically compose, deploy and query software components in ubicomp environments. It uses an annotated version of ReWiRe's behaviour model that links different rules together. The annotations are then processed at runtime to build up a model of the system's behaviour that can be easily queried and is used to generate the list of why and why not questions. It features displays that are equipped with a motion sensor to detect the presence of the user.

We illustrate how our approach works by means of an example scenario, shown in Figure 2. We will follow Bob, one of the visitors of the smart museum equipped with PervasiveCrystal. When he enters the museum, Bob receives a mobile museum guide that can be used to interrogate and control the environment. Bob is told that the museum features displays that can detect his presence and react to motion.


motion sensor
movement
start movie
movie is playing
lights out
turn off lights
Why did the lights go out?
Two rules interacted with each other.
A
Questions
Why?
Why not?
did the lights go out? (a few seconds ago)
did a movie start playing? (a few seconds ago)
Why did the lights go out?
B
4
Light:
Roman Art Room
100
ON
1
2
3
Turn lights on
Light control UI
Close

Why did the lights go out?

The lights did go out because a movie started playing.

The movie started playing because motion was sensed.

Figure 2: PervasiveCrystal shows a list of available questions, based on recent events (*A*). Answers are generated by linking events to what caused them to happen (*B.1*). Additionally users have access to two control mechanisms: they can *undo* operations (*B.2*) or invoke fine-grained control user interfaces, in this case: a light control UI (*B.4*).

When Bob approaches one of these displays during his museum visit, he waves in front of the screen to play a movie, as shown in Figure 2 (scene 1). However, at that time, the lights also go out. Bob does not understand why this happens, and is confused (scene 2). Behind the scenes, there are several rules that react to context changes (scene 3). One of the rules plays a movie when the camera detects motion. There is also another rule that turns off the lights whenever a movie is playing to provide users with a better viewing experience. When the first rule executes, its effect (playing a movie) causes the second rule to execute and turn off the lights. Bob remembers he can use the why menu to ask questions about the smart museum's behaviour (scene 4). As seen in Figure 2 (4.A), the why menu shows a list of available questions about events together with a representative icon. PervasiveCrystal automatically generates the list of questions by tracking events that occurred (e.g., lights that are switched off).

The Visible Computer

Because of the heterogeneous nature of ubicomp environments — which often employ several displays, speakers, sensors, embedded devices — users might require co-located information that tells them what the system is doing, *when* and *where* it is doing this, and allows them to intervene without leaving their current task.

To explore this idea, we developed a prototype [24] that uses steerable projectors to overlay the environment with real-time visualizations of actions occurring in the environment (*e.g.* turning off the lights). Figure 3 shows how we used a simple graphical language to visualize the relationships between sensors or devices and system actions. When an action is executed, an animation is shown to visualize the cause(s) and consequence(s) of this action. In addition, users can issue a voice command to cancel (or *undo*) the most recent action.

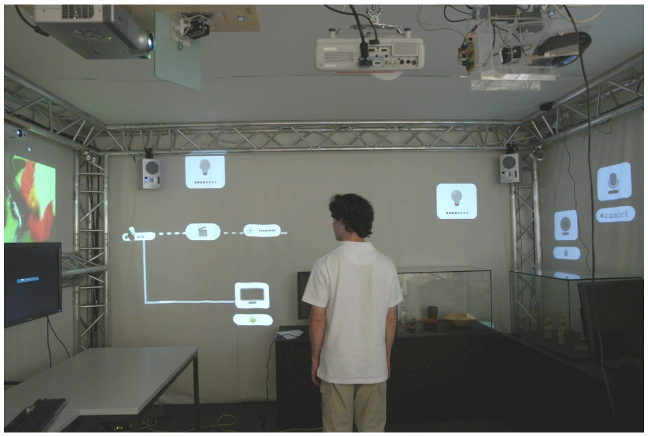


Figure 3: A user looks at an animation that links sensors and devices with system actions to explain the system's behavior.

Each sensor or input/output device (e.g., a camera, speaker, display) is visualised at its physical location in the environment with an icon and a label. These icons allow users to get an overview of the devices that are present in their environment. Below the icon of each *input* device or sensor, a separate label is drawn that displays the possibilities of the device and its current state using smaller icons. *Output* devices feature only an icon and no separate label. The icon of an output device embeds its current state (e.g., a light's intensity displayed as a horizontal bar. Figure 4 shows how a chain of events is visualised using this graphical language.

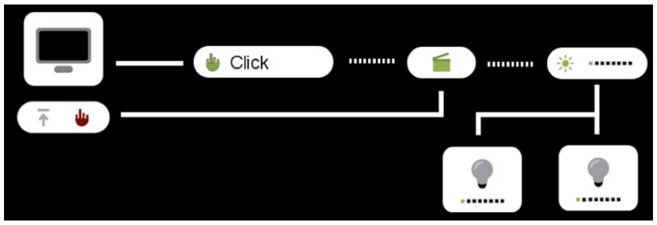


Figure 4: Visualising a chain of events: touching the screen results in a movie being played. This, in turn, results in the lights being dimmed.

The Feedforward Torch

We have built the Feedforward Torch [25], a prototype to explore *feedforward*. Feedforward is a specific type of intelligibility information that informs the user about *what the result of an action will be* [23]. If we consider the timing dimension, feedforward is thus intelligibility information that is provided *before* an event takes place. While feedback tells the user what happened, feedforward tells the user what will happen. Well-designed feedforward is an effective tool for bridging Norman's Gulf of Execution [18] – the gap between a user's goals for action and the means for executing those goals [23]. Ju et al. [13] also talk about feedforward as a specific variation of their user reflection technique.

The Feedforward Torch is a combination of a smartphone and mobile projector that provides feedforward about the objects and interactions in the user's environment. With the Feedforward Torch, we do not focus exclusively on ubicomp environments, but also target existing legacy systems in our daily environments. We argue that these environments require intelligibility as well. If users have difficulties interacting with the system, having to fix this after deployment is very cumbersome and expensive. Physically changing the interface design to include better feedforward would imply fixing every instance of the system separately. The Feedforward Torch augments the systems during usage and does not require physical changes in order to overcome design flaws of legacy systems.

Users can point the Feedforward Torch at objects in their environment and reveal feedforward information about them, as if they were located under a spotlight. Users are shown under which conditions actions associated with the object will be executed by the system (e.g., a displacement in time or space), so that they can anticipate and adapt their behaviour, if necessary. Animations are used to better convey the effect an action will have. The Feedforward

Torch does not extend the features of a legacy system; its sole focus is on guiding the user to use the actual system.

The main difference between the Feedforward Torch and the Visible Computer, which used steerable projectors, is the fact that the Feedforward Torch places the *initiative* for showing information with the *user*. Figure 5 shows how the Feedforward Torch can be used to understand an array of light switches. Like in the Visible Computer prototype, information is projected on and around the system, so that users do not need to switch their attention to another interface and can continue to focus on the task they are performing (*co-location*: *embedded*).

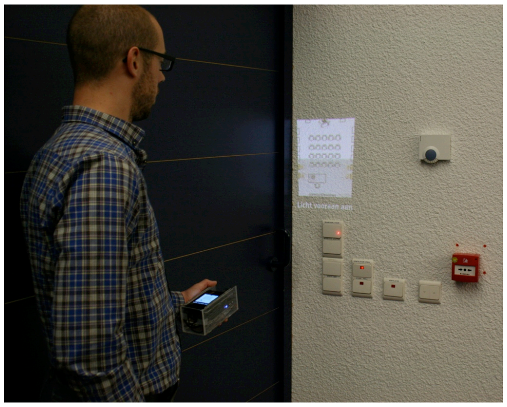


Figure 5: A user points at a light switch using the Feedforward Torch to understand what will happen if he presses the switch.

Figure 6 shows the Feedforward Torch prototype, consisting of a Samsung Galaxy S smart phone, a MicroVision SHOWWX+ laser pico projector and a laser pointer to be able to point the device at physical objects. A custom casing was made in order to support one-handed interaction. We used a Wizard-of-Oz control interface to show the right content to the user at the right time in order to simulate a fully working object recognition mechanism.

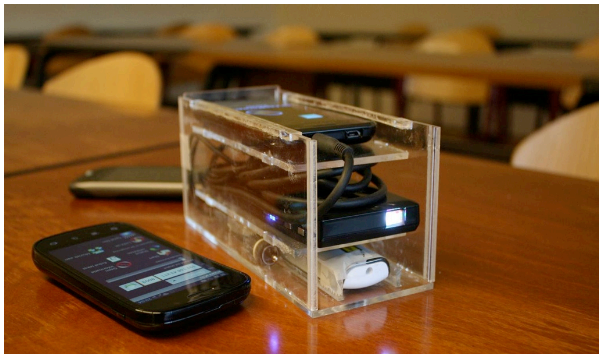


Figure 6: The Feedforward Torch prototype (right) and Wizard-of-Oz control interface (left).

MAPPING THE DESIGN SPACE

In Figure 7, we show how the different systems that were discussed in this paper fit into the proposed design space. We will discuss each of the six dimensions.

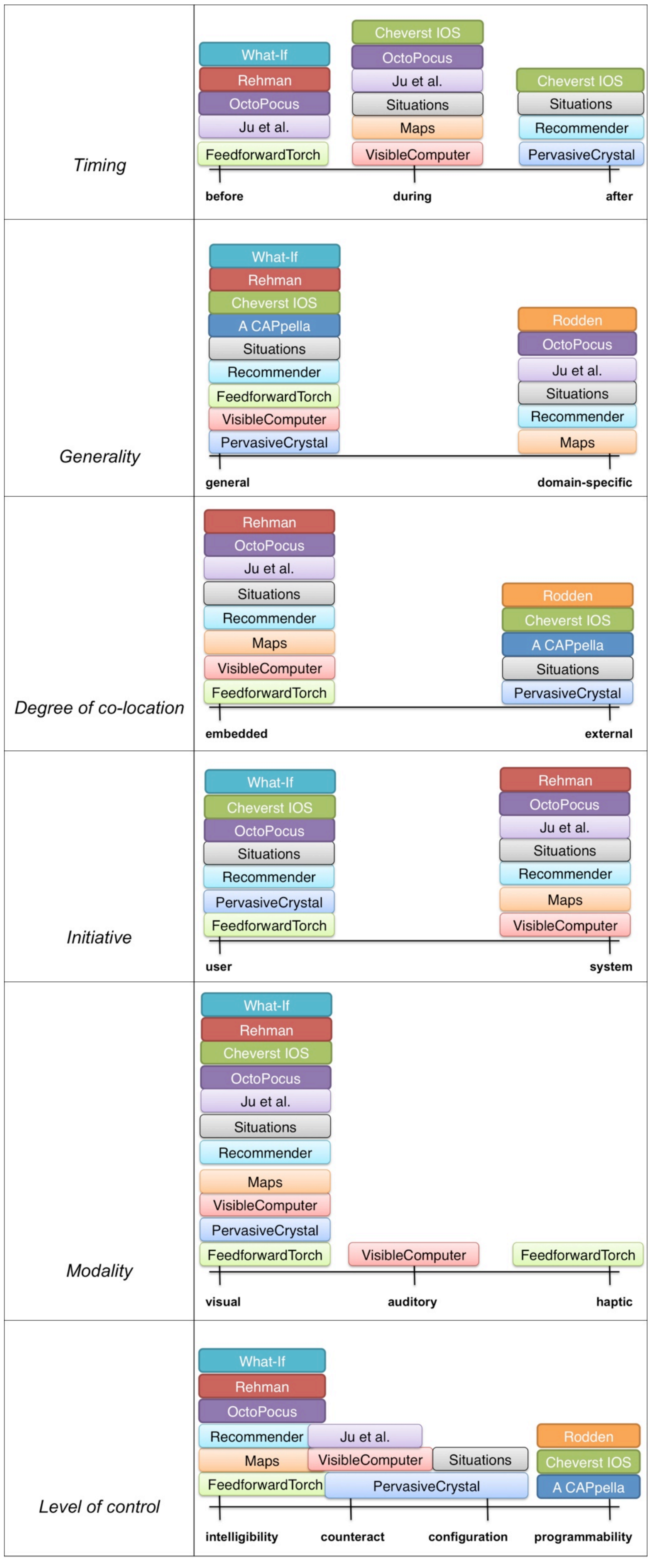


Figure 7: Mapping the different system or techniques in the design space for intelligibility and control

Timing: With respect to timing, there is quite some diversity. There are a few techniques such as those of Ju et al. [13], OctoPocus [3] or Cheverst's IOS system [5] that span multiple alternatives of the timing dimension, but most techniques only offer a specific moment in time at which intelligibility is provided. Of course, ideally, systems should be intelligible about past, present and future events.

Generality: We covered a variety of domain-specific and general interfaces in our design space. Some systems or techniques can provide both domain-specific and general intelligibility, e.g., recommender systems, what if questions and Situations [10].

Degree of co-location: We notice that most domain-specific interfaces are also embedded (e.g., the location error visualisation in Google Maps). However, this is not always the case. For example, Rodden's jigsaw editor [20] is a domain-specific interface for controlling a smart home, but is nevertheless external.

Initiative: Most systems either provide intelligibility automatically, or allow the user to request detailed information when necessary. As discussed earlier, some systems combine both approaches to support a flexible transition from novice users to experts, such as OctoPocus [3].

Modality: It is apparent that most techniques rely on the visual *modality*. There are only a few systems that provide intelligibility through other means, and even then, these are providing visual information as well (e.g., the Visible Computer and the Feedforward Torch).

Level of control: There are a number of systems that only provide intelligibility (e.g., Google Maps, the Feedforward Torch, OctoPocus) without any control mechanism. On the other end of the spectrum, there are very powerful systems such as Rodden's jigsaw editor [20], IOS [5] and a CAPpella [8]. However, these techniques usually employ an external interface and are very general. One could argue whether these techniques are really usable by non-technical users.

LESSONS LEARNED

In what follows, we reflect on our experiences from conducting first use studies with the different systems discussed earlier (PervasiveCrystal, the Visible Computer, and the Feedforward Torch). We discuss the lessons learned with respect to the previously introduced design space.

PervasiveCrystal

We learned that textual explanations are not always ideal. Automatically generated explanations can sometimes be confusing to users, especially when they describe a long sequence of events that caused a system action, or when double negations are involved (e.g., "The lights didn't go out because the movie didn't start playing"). While there are a number of strategies to overcome this problem (simplifying or combining several explanations), we believe some situations might be too complex to explain solely using text.

Another problem users faced is that the why questions menu quickly became cluttered due to many events firing in a short time span. This made it hard for users to find the question they wanted to ask. Unlike desktop applications that typically use explicit interaction, ubicomp environments use implicit interaction and sensors that trigger many events (e.g., a motion sensor). While these questions could be clustered, it might still be hard for users to find the question about the event that they are interested in. With respect to the control primitives, participants also found it hard to predict the effect of invoking undo and do, after which we used more concrete labels (e.g., "Turn on lights" instead of "Undo"). The fact that the effect of the undo and do actions was hard to predict might also be due to the *external* nature of the interface. Why questions that are *embedded* into specific applications (e.g., the word processor built with Crystal [17]) might be less disconnected from the user's task.

The Visible Computer

We ran an informal study with five participants to gather feedback about the suitability of our approach of visualizing the system's behaviour using steerable projectors. Subjects were asked to explain how the system worked in three different situations, after having seen the visualization. Four out of five subjects could describe the system's behaviour correctly for each of the three situations. This promising result could indicate that a visual, *embedded* way of presenting how the system works might help users to form a better mental model, which is consistent with findings by Rehman et al. [19]. Participants were generally happy with the visual representation, but sometimes had difficulties with keeping track of visualizations across multiple surfaces. One participant mentioned she received too much information, leaving her overwhelmed. This might indicate that we should be careful when automatically providing detailed explanations (*system initiative*). Finally, several subjects experienced difficulties with invoking the cancel feature, possibly because they were not familiar with speech interaction (*modality*: *auditory*).

The Feedforward Torch

We also conducted a small study with the Feedforward Torch. We used a Wizard-of-Oz control interface to change the contents of the feedforward display.

All participants were able to complete the tasks and several participants mentioned they would have been unable to do so without the Feedforward Torch or additional help. Two participants stated that the system would have come in handy in a large city: "*When I had to use the London Underground for the first time, the Feedforward Torch would have been useful to help me figure out how to use the ticketing machine. Now, I had to observe other passengers first before I knew how the system worked and what I had to do.*"

Participants liked the fact that information was overlaid on the physical environment (*embedded*), so they did not have to switch between the smartphone display and the system or device they had to operate. One of the advantages of mobile projection that was mentioned during the semi-structured interviews was the fact that groups of people could explore the projection together. Nevertheless, projection only worked well in low-lighting conditions. The use of animations was appreciated, especially when the result of a certain action would happen over time or outside the user's periphery. Finally, participants strongly preferred visualisations to textual explanations in the encountered scenarios, as they considered reading textual information to be more time-consuming.

We can conclude that the choice between different combinations of each of these dimensions for intelligibility and control interfaces can have a large impact on the user experience. Designers can use our design space to consider these different alternatives and choose the one that fits their application best.

REFERENCES

1. Aksenov, P., Luyten, K., and Coninx, K. O Brother, Where Art Thou Located? Raising Awareness of Variability in Location Tracking for Users of Location-based Pervasive Applications. *J. Location Based Services*, 6 (4), (2012), 211-233.
2. Barkhuus, L. and Dey, A.K. Is Context-Aware Computing Taking Control away from the User? Three Levels of Interactivity Examined. *Proc. Ubicomp '03*, Springer (2003), 149–156.
3. Bau, O., and Mackay, W. E. OctoPocus: a dynamic guide for learning gesture-based command sets. *Proc. UIST '08*, ACM (2008), 37–46.
4. Bellotti, V. and Edwards, W.K. Intelligibility and accountability: human considerations in context-aware systems. *Hum.-Comput. Interact. 16*, 2 (2001), 193–212.
5. Cheverst, K., Byun, H.E., Fitton, D., Sas, C., Kray, C., and Villar, N. Exploring Issues of User Model Transparency and Proactive Behaviour in an Office Environment Control System. *User Modeling and User-Adapted Interaction 15*, 3–4 (2005), 235–273.
6. Cheverst, K., Davies, N., Mitchell, K., and Efstratiou, C. Using Context as a Crystal Ball: Rewards and Pitfalls. *Personal Ubiquitous Comput. 5*, 1 (2001), 8–11.
7. Coutaz, J. Meta-User Interfaces for Ambient Spaces. *Proc. TAMODIA '06*, (2006), 1-15.
8. Dey, A.K., Hamid, R., Beckmann, C., Li, I., and Hsu, D. a CAPpella: programming by demonstration of context-aware applications. *Proc. CHI '04*, ACM (2004), 33–40.
9. Dey, A.K. Understanding and Using Context. *Personal Ubiquitous Comput. 5*, 1 (2001), 4–7.
10. Dey, A.K. and Newberger, A. Support for Context Intelligibility and Control. *Proc. CHI '09*, ACM (2009).
11. Dourish, P. Accounting for system behavior: representation, reflection, and resourceful action. In *Computers and design in context*, MIT Press (1997), 145–170.
12. Edwards, W.K. and Grinter, R.E. At Home with Ubiquitous Computing: Seven Challenges. *Proc. UbiComp '01*, Springer-Verlag (2001), 256–272.
13. Ju, Lee, and Klemmer. Range: exploring implicit interaction through electronic whiteboard design. *Proc. CSCW '08*, ACM (2008), 17–26.
14. Lim, B.Y. and Dey, A.K. Assessing Demand for Intelligibility in Context-Aware Applications. *Proc. Ubicomp '09*, ACM (2009), 195–204.
15. Lim, B.Y., Dey, A.K., and Avrahami, D. Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. *Proc. CHI '09*, ACM (2009), 2119–2128.
16. Lim, B.Y., Dey, A.K. Toolkit to Support Intelligibility in Context-Aware Applications. Proc. Ubicomp '10, ACM (2010), 13–22.
17. Myers, B.A., Weitzman, D.A, Ko, A.J., and Chau, D.H. Answering why and why not questions in user interfaces. *Proc. CHI '06*, ACM (2006) 397–406.
18. Norman, D. A. *The Psychology Of Everyday Things*. Basic Books, New York, USA, June 1988.
19. Rehman, K., Stajano, F., and Coulouris, G. Visually Interactive Location-Aware Computing. *Proc. Ubicomp '05.*, (2005), 177–194.
20. Rodden, T., Crabtree, A., Hemmings, T., et al. Configuring the Ubiquitous Home. *Proc. COOP '04*, (2004), 227–242.
21. Schmidt, A. Implicit Human-Computer Interaction Through Context. *Personal Ubiquitous Comput.* 4, 2/3 (2000), 191–199.
22. Vanderhulst, G., Luyten, K., and Coninx, K. ReWiRe: Creating interactive pervasive systems that cope with changing environments by rewiring. *Proc. IE '08*, (2008), 1–8.
23. Vermeulen, J., Luyten, K, van den Hoven, E., Coninx, K. Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. *Proc. CHI '13*, ACM (2013), 1931–1940.
24. Vermeulen, J., Slenders, J., Luyten, K., and Coninx, K. I Bet You Look Good on the Wall: Making the Invisible Computer Visible. *Proc. AmI '09*, Springer-Verlag (2009), 196–205.
25. Vermeulen, J., Luyten, K., and Coninx, K. Understanding Complex Environments with the Feedforward Torch. *Proc. AmI '12*, Springer-Verlag (2012), 312–319.
26. Vermeulen, J., Vanderhulst, G., Luyten, K., and Coninx, K. PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. *Proc. IE '10*, (2010), 271–276.