PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications

Jo Vermeulen Geert Vanderhulst Kris Luyten Karin Coninx
Hasselt University – tUL – IBBT
Expertise Centre for Digital Media,
Wetenschapspark 2, 3590 Diepenbeek, Belgium.
Email:{jo.vermeulen,geert.vanderhulst,kris.luyten,karin.coninx}@uhasselt.be

***Abstract*—Users often become frustrated when they are unable to understand and control a pervasive computing environment. Previous studies have shown that allowing users to pose why and why not questions about context-aware applications resulted in better understanding and stronger feelings of trust. Although why and why not questions have been used before to aid in debugging and to clarify graphical user interfaces, it is currently not clear how they can be integrated into pervasive computing systems. We explain in detail how we have extended an existing pervasive computing framework with support for why and why not questions. This resulted in PervasiveCrystal, a system for asking and answering why and why not questions in pervasive computing environments.**

I. Introduction

Pervasive computing systems are generally *context-aware*, which means that they act based on *context* [6] – implicit input collected from the environment. Since these systems often act without explicitly involving the user, users may be 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 understand [9].

However, being difficult to understand is only part of the problem. Context-aware systems have been shown not to be infallible. They are bound to sometimes make mistakes because of the inevitable incompleteness of context information [5], [3], [7]. 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 result in them losing trust in the system [2].

Bellotti and Edwards [3] proposed two design principles to tackle these problems: *intelligibility* (also known as *scrutability* [4]) 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* to recover from possible mistakes.

One way to improve the intelligibility of a context-aware system is to allow users to pose *why and why not questions* about its behaviour. Recently, a number of studies [13], [12] have suggested that supporting these questions would result in better understanding and stronger feelings of trust. By asking *why* and *why not* questions, arising respectively from *unexpected* events that *occurred* or *expected* events that *did not occur*, users gain a better understanding of the internal working of the system they interrogate. However, there is to date no pervasive computing framework available that supports why and why not questions. Moreover, existing desktop implementations (e.g. Crystal [14]) cannot be easily integrated into pervasive computing frameworks, since the assumptions underlying these implementations (e.g. having a single machine from which events originate) rarely hold in pervasive computing.

In this paper, we describe how we have extended the *ReWiRe* pervasive computing framework [15] to allow users to pose why and why not questions about occurring events, resulting in the creation of PervasiveCrystal. We explain how PervasiveCrystal works by means of an example scenario (Sect. III). The cornerstone of the PervasiveCrystal extension is the behaviour model which allows us to trace events across distributed components and which is easy to query (Sect. IV). We use the behaviour model to generate possible why (not) questions (Sect. V-A) and provide answers to these questions (Sect. V-B). The behaviour model also allows us to provide users with three three simple *control* mechanisms: undo; do; and task-specific control user interfaces (Sect. V-C). We discuss the results of a pilot user study (Sect. VI), give an overview of PervasiveCrystal's limitations, and outline opportunities for future work (Sect. VII).

II. Related Work

Existing work has demonstrated the potential of allowing users to ask questions about the behaviour of their software. This approach has been successful in decreasing the time programmers spend debugging their programs [10], [11] and in improving end-user understanding and control in desktop applications [14]. Crystal [14] is an application framework that provides developers with an architecture and a set of interaction techniques to build desktop applications that answer why questions. It is targeted at explaining complex behaviours and interdependencies

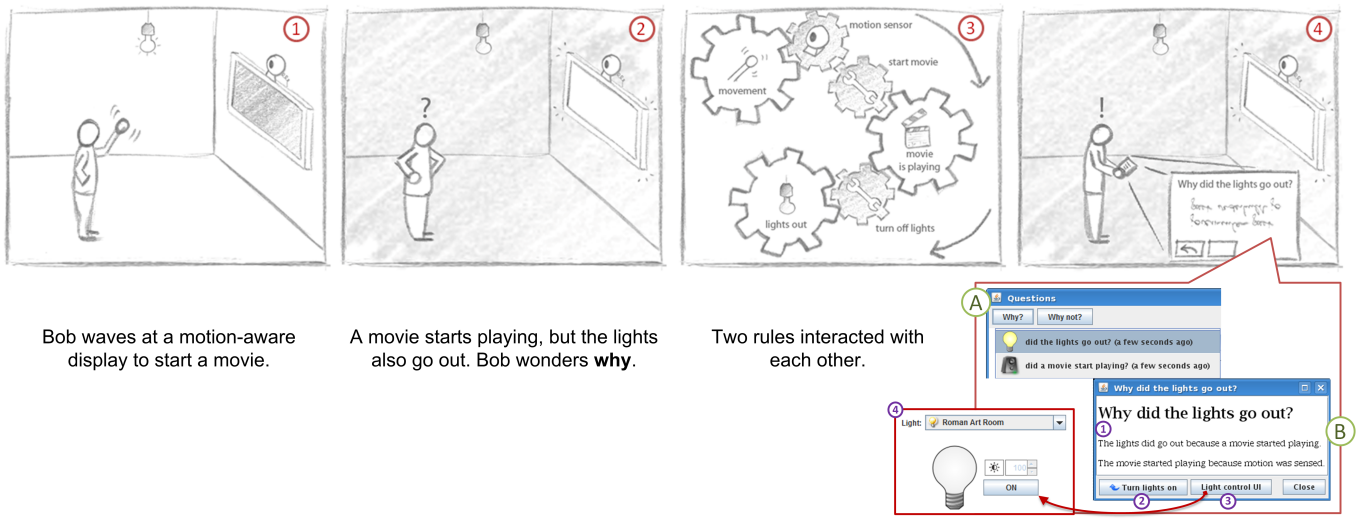


Figure 1. Posing a *why* question: PervasiveCrystal shows a list of available questions, based on events that recently took place in the environment (*4.A*). Answers are generated by linking events to what caused them to happen (*4.B.1*). Additionally, users have two means for correcting the environment's behaviour: they can *undo* operations (*4.B.2*) or invoke fine-grained control user interfaces (*4.B.3*), in this case: a light control user interface (*4.B.4*).

among the various features of an application to *end-users*. Crystal's interaction techniques not only improve end-users' understanding of the software, but also help users in determining how they can fix unwanted behaviour.

Lim, Dey and Avrahami [13] investigated if why (not) questions could be used to improve understanding 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 the study, a comparison was made between different types of questions users could ask about a context-aware system. *Why* and *why not* questions were found to be the most effective, as opposed to *what if* and *how to* questions that did not contribute much to users' understanding of the system or their perception of trust. In a later study, Lim and Dey investigated the different information demands users have for context-aware applications under various situations [12]. They recommend that *why* questions should be made available for all context-aware applications, while *why not* questions are more useful for specific contexts (e.g. goal-supportive tasks, high risk tasks).

Although previous work suggests that it is useful to provide support for why and why not questions, there is to date no pervasive computing framework available that makes it easy to build applications that can answer these questions. Unfortunately, developers cannot rely on existing implementations for different target domains either, since these are often based on underlying assumptions that do not hold in pervasive computing. Crystal [14], for example, assumes that it runs on a single machine and that it has access to an in-memory tree of command objects. In pervasive computing, however, applications are often distributed over several devices, and events can originate from different networked components (e.g. sensor nodes), making it challenging to construct correct cause-effect chains.

III. Usage Scenario

We illustrate how our approach works in practice by means of an example scenario. In this walkthrough, we will follow Bob, one of the visitors of a smart museum equipped with PervasiveCrystal. As Bob enters, he receives a mobile museum guide that can be used to interrogate and control the environment.

A. Why Questions

Bob was told that the museum features interactive screens that react to motion. When Bob approaches one of these displays during his visit, he waves in front of the screen to play a movie, as shown in Fig. 1 (*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, the system uses a rule-based system to react to context changes (*scene 3*). One of the rules plays a movie when motion is detected by a camera. 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 Fig. 1 (*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 questions are presented in reverse chronological order (questions about the most

recent events come first). Bob then selects the question "Why did the lights go out?", and receives an answer that briefly explains what caused both rules to fire (*4.B*). PervasiveCrystal can generate these answers by linking events to what caused them to happen. In this case, the system knows that the lights went out because a movie started playing. When enough information is available, the system will explain the entire execution trace of the event (*4.B.1*). Here, the system also includes an explanation of why the movie started playing in its answer: "because motion was sensed". Explanations for chains of interacting rules can be of arbitrary length.

Besides helping Bob to understand why the system has taken a certain action, PervasiveCrystal also allows Bob to intervene and correct unwanted behaviour. Within answer dialogs, such as the one of *scene 4.B* in Fig. 1, users have two ways of controlling the system. First, the left button allows users to *undo* unwanted actions (*4.B.2*). In Bob's case, clicking the button will turn the lights back on again, thereby undoing the action taken by the system. Secondly, PervasiveCrystal provides users with more fine-grained control user interfaces to correct undesired behaviour. The second button from the left (*4.B.3*), allows users to invoke a task-specific control user interface. Here, Bob can bring up the light control user interface, providing him with more options such as the specific intensity of each of the lights in the museum (*4.B.4*). PervasiveCrystal achieves this by annotating events with related user tasks (e.g. controlling lights, playing media) and their respective user interfaces.

B. Why Not Questions

Later, Bob returns to the display and tries to start the movie again, as shown in Fig. 2 (*scene 1*). However, this time, nothing happens. Bob does not understand why the system acts differently now (*scene 2*). Behind the scenes, the camera motion sensor never reported that it detected motion and as a result, the rule that is responsible for playing the movie never got executed (*scene 3*). Bob then remembers that he can also pose *why not* questions. For this, Bob uses the *why not menu* (see Fig. 1 *4.A*), which works in a similar way as the why menu that was discussed in Sect. III-A. However, instead of listing questions about events that did occur, the why not menu presents users with a list of questions about expected events that *did not occur*.

When expected events do not take place, the cause is often an unexecuted rule which was supposed to trigger the event. PervasiveCrystal keeps track of which rules can trigger which events, and analyses unexecuted rules to fill the why not menu with a list of questions about events could have taken place (but did not). Based on the available information about a rule, it then tries to determine *why* these rules did not execute.

Bob proceeds by selecting the appropriate question in the why not menu: "Why didn't the movie play?" (*scene 4*). The system responds by saying that no motion was sensed (*4.A*). Bob then figures that something must be wrong with the motion detection, and notices that the camera cable is unplugged.

The why not answer dialogs again provide users with two means for controlling the system behaviour. Just as with why questions, users can invoke a fine-grained control user interface (*4.C-4.D*). The undo command, however, has been replaced by a *do* command that allows users to force the system to execute an action (*4.B*). *Undo* is available for *why* questions, while *do* is available for *why not* questions. Bob decides that he wants to see the movie anyway, and forces the system to *do* the operation anyway by clicking the "Play movie" button (*4.B*).

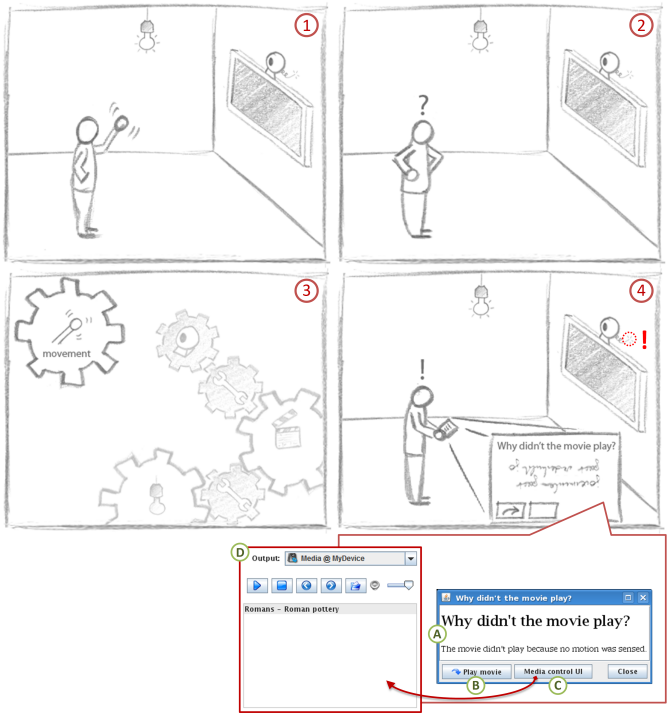
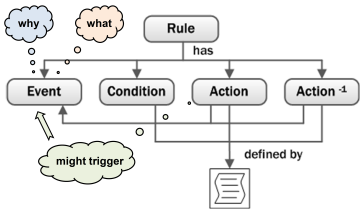


Figure 2. Posing a *why not* question: This time, nothing happens when Bob moves in front of the display. By asking a why not question, Bob is able to figure out that the system did not sense motion (*4.A*). He then notices that the camera cable is unplugged. Bob is again provided with different ways to control the environment. He can use the *do* command to force the system to play the movie anyway (*4.B*), or bring up the media control user interface (*4.C-4.D*).

IV. Behaviour Model

We capture the behaviour of an environment in a model built up from rules that connect actions and events. An action is caused either by an end-user who is interacting with the environment or by the system itself, for example as a reaction to a context change. A change in the environment's configuration results in an event which in turn can cause the system to execute new actions as specified by the rules currently defined in the model. These rules are implemented according to the Event-Condition-Action (*ECA*) paradigm [1], and are extended with inverse

actions ($ECAA^{-1}$) as shown in Fig. 3(a). Rules with inverse actions can be undone, making it possible to return to a former state. This is achieved by caching the execution context of a rule's action (i.e. environment properties relevant for the rule) and passing this context as input to a rule's inverse action.



(a) An annotated $ECAA^{-1}$ rule.

```
1  importPackage(Packages.environment.behavior.model);
2  importPackage(Packages.services.light.sensors);
3
4  var evt = new BEvent("*", MoviePlayingSensor.URI,
5                       "Movie started playing");
6  var act = new BAction("turnOff", "Turn light off");
7  act.addMightTrigger("*", LightOffSensor.URI);
8
9  proc.addRule(new BRule(evt, act));
10
11 this.turnOff = function() {
12     alert("Turning off light");
```

(b) Editor for scripting behaviour rules.

Figure 3. The behaviour model is composed of annotated $ECAA^{-1}$ rules. Rules can be created and annotated in code using ReWiRe's behaviour script editor.

To generate accurate and complete explanations of why the system behaves in a certain way, we adopt the behaviour model in both scripts and control user interfaces. Scripts enable developers to quickly prototype and customize the overall behaviour of the environment while control user interfaces focus on end-users and specific application domains (e.g. Fig. 1 *(4.B.4)*: a user interface for controlling the different lights in the environment).

Fig. 3(b) shows how behaviour rules can be defined and added to the model with ReWiRe's script editor. This specific script creates the rule from the museum scenario that turns off the lights when a movie is playing (see Sect. III). Scripts allow developers to change the environment's behaviour in just a few lines of JavaScript.

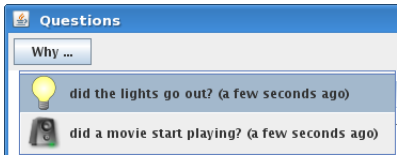
Additionally, control user interfaces create underlying behaviour rules so that the system can track what happens in these user interfaces as well (e.g. explaining that the lights were turned off because the user did this in the lights control user interface). Dey and Newberger took a similar approach with Situations [8] which allows designers to provide domain-specific intelligibility and control user interfaces. We support why (not) questions as follows:

- *Logging user actions*: Control user interfaces are programmed to log every user action that occurs and optionally define inverse actions to undo unwanted side effects. Since a mix of user-driven and system-driven behaviour is typical for a pervasive computing environment, past user actions help to provide insight in the current state of the environment.
- *Semantic annotations*: We annotate behaviour rules with additional information about the events and actions they might trigger (see the bubbles in Fig. 3(a) and line 4-7 of Fig. 3(b)). By analysing and simulating the execution of rules, we can predict what might happen when an event occurs or when the user executes a certain action. Although we are dealing with uncertainty when simulating rules – a rule's condition might evaluate differently when a rule is effectively executed due to changes in the environment – a better understanding of the pervasive computing system can be achieved. Moreover, annotations help to filter the events and actions that gave rise to the behaviour that is questioned by the user. The chain of events and actions is crucial information for accurate answers to why and why not questions.

V. Allowing Users to Understand and Intervene

A. Generating Questions

Fig. 4(a) shows how generated questions about events are listed in a pop-up menu, together with a representative icon, and are sorted from on the time at which they occurred (most recent first). When a user selects one of the questions, an answer dialog is shown, as seen in Fig. 4(b). To make this possible, all events, actions and conditions are expected to have a short descriptive label. Events that affect end-users, are enriched with *why* and *what* descriptions (see Fig. 3(a)). These are plain text strings in which grammatical constructs (e.g. auxiliary verbs) have been annotated to be able to negate the question in case of a *why not* form. The *why* descriptions are used to generate questions for events in the why menu. Only questions for events with why and what descriptions are shown in the



(a) The why menu.

(b) An answer to a why question.

Figure 4. The why menu allows users to pose why and why not questions about the things that happen in the environment. Users receive answers to their questions, and are offered a means to recover from undesired behaviour.

menu. Other questions are assumed to only be of use to the system, and will not be of interest to end-users.

B. Generating Answers

Answers to questions mainly depend on *what* descriptions and annotated ECA rules. In the scenario of Sect. III, Bob poses the question "Why did the lights go out?" and is shown a corresponding explanation dialog (Fig. 4(b)). The *what description* of this particular event would then be `the lights did go out`. This description is used by the system to provide the first part of the answer, as shown in Fig. 4(b). The rest of the answer is automatically generated from analyzing the ECA rules. The event about which Bob asks a question is traced back to the ECA rule that caused it to happen (see Fig. 1 *scene 3*). PervasiveCrystal looks for the responsible rule by querying the behaviour model for actions that have a `mightTrigger` annotation for the event the user asks a question about (see Fig. 3(a)), and which were recently executed. When the system knows which rule is responsible, it completes the answer by explaining which event caused the responsible rule to execute. The responsible event's *what* description (`a movie started playing` in this case) is added to the answer, as seen in Fig. 4(b). When the system notices that the responsible event was again the effect of another rule, it repeats the process for that rule and adds the corresponding answer below the first answer. In Fig. 4(b), the system adds a new paragraph with the explanation of why the movie started playing. When an executed rule required a condition to be true, this condition is also added to the explanation using its short descriptive label. If the chain of ECA rules eventually traces back to a user action (e.g. in a control user interface), the explanation would describe this with "because you did ...".

Answers to *why not* questions are generated in a similar way, but are are more difficult to trace. PervasiveCrystal currently supports *why not* questions in two ways. First, why not questions are possible about events resulting from rules which fired but did not execute because their condition was false. The corresponding answer will then explain that the event did not occur because the condition was false. Secondly, we allow why not questions about events resulting from rules which never executed because their triggering event never fired. This kind of why not question is posed in Fig. 2 (*scene 4*). There is typically more uncertainty in answering why not questions than in answering why questions. Since a why not question asks about an event which never occurred, the system cannot rely on tracing and has to reason about what could happen. It is important to ask about why not questions about events within a certain time frame, to avoid overwhelming the user with possible answers. Currently, we use a brute force approach that enumerates all possible causes. The next step is to include a better estimation of timeliness (related with the type of event and condition) and the likelihood of the candidate answers.

C. Giving Users Control

When users have understood why the system acted in a certain way, they can choose to intervene and correct the system, if necessary. The explicit *undo* operation is supported by calling the inverse action of an $ECAA^{-1}$ rule (see Sect. IV and Fig. 3(a)). *Do* will just execute the action of an ECA rule, regardless of the event or condition. Besides explicit *undo* and *do* operations, we also provide more fine-grained control user interfaces to correct the behaviour of the system. As shown in Fig. 4(b), there is a second button allowing Bob to invoke the light control user interface. This is achieved by further annotating an action with the different user goals the action can contribute to (e.g. controlling lights, playing media).

VI. Preliminary User Study

A. Participants and Method

We conducted a pilot user study with a first iteration of PervasiveCrystal [17] to get an idea of its ease of use. We asked five volunteers (4 male, 1 female) to use our system to understand and control the behaviour of a pervasive computing environment in different situations. Four out of five participants had programming experience, while the fifth participant had a background in social sciences. The experiment was carried out in a realistic pervasive computing environment: an interactive museum room which next to museum artefacts also features different kinds of sensors, and various means to provide visitors with information. Subjects used a networked Ultra-Mobile PC (a Samsung Q1 Ultra running Windows XP and the PervasiveCrystal client software) to ask why questions and view the corresponding answers.

Participants were presented with three situations in which something happened that they had to explain and control using PervasiveCrystal. In the first task, participants were told to sign in to the system after which a song would start playing. For the second task, participants were asked to go and stand in front of a display that used a webcam to perform motion detection as an indicator of the user's presence. When subjects would walk up to the display, a movie would start playing. The third and final task was similar to the second one, but here the movie would only play while motion was detected and immediately stop otherwise. Participants were asked to try to understand what was different compared to the previous situation. Additionally, they had to use one of our control mechanisms to find a way to keep playing the movie without having to move in front of the display all the time. Finally, we conducted a semi-structured interview in which participants were asked to comment on the features of our system.

B. Results

All subjects were able to use the questions interface to find the cause of events in these three tasks. Overall, participants found that the answers to the why and why

not questions were what they wanted to know, although most participants argued that the way they were presented could be improved. Each participant agreed strongly on the fact that these techniques are useful to allow users to understand what happens in their environment and offer them control over this behaviour.

One of the major problems users faced was the fact that the why-menu quickly became cluttered when many events were firing in a short time span. This made it hard for subjects to find the question they wanted to ask. Especially in the third task, users would several times trigger the motion sensor, causing a clutter of why questions in the user interface. There are a number of ways to overcome this problem: users could be offered a way to filter events (e.g. only questions about music or video), or events that occur very often in a short time period could be clustered in one why question (e.g. "Why did a movie stop playing (10x in the last 20 seconds)").

Three out of five subjects were able to successfully use our control mechanisms to achieve the desired effects. The remaining two participants were given a few cues, but did not require much assistance to complete the tasks either. The majority of participants used the fine-grained control user interfaces. Subjects found this mechanism useful and could quickly figure out how to use it to control the environment. However, subjects were less positive about the ease of use of our *undo* and *do* commands. In the first iteration, we used generic labels ("undo" and "do") for these commands, which made it hard for users to predict their effect. Based on this observation, we later changed our implementation to use more specific labels for the undo and do buttons based on the label for the corresponding action – such as "Stop video" and "Turn on lights".

VII. Discussion

This paper explained how an existing pervasive computing framework was extended with support for *why and why not questions*. This requires a behaviour model that can easily be queried to reason about the environment's behaviour. Besides being a feasibility demonstration, we believe that PervasiveCrystal can help developers to create pervasive applications that are more usable, predictable and safe for end-users [3].

Our current implementation has a few limitations. First, it is yet unclear whether the system is scalable to large and complex applications. Especially why not questions pose scalability challenges since the number of possibilities that have to be examined for these questions rises exponentially when the complexity of the environment increases. It will be necessary to find a balance between adequate memory and performance on the one hand, and sufficiently accurate information on the other hand. Nevertheless, we believe the biggest challenge lies not in optimizing CPU or memory usage, but in providing users with detailed and complete information about the system's behaviour without overwhelming them. Second, PervasiveCrystal can only handle rule-based systems. It was not designed to deal with machine learning algorithms (e.g. decision trees, neural networks). Finally, we assume that the annotations that developers provide are correct and consistent with the actual behaviour of the application.

In future work, we would like to conduct a larger user study to evaluate the questions interface. Moreover, we are looking into improving support for why not questions and further reducing the required developer effort in providing annotations for the questions. Finally, as textual explanations might not be suitable in all situations [12], we are also exploring ways to provide a graphical representation of the environment's behaviour [16].

References

[1] C. Act-Net Consortium. The active database management system manifesto: a rulebase of adbms features. *SIGMOD Rec.*, 25(3):40–49, 1996.

[2] L. Barkhuus and A. K. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. In *Proc. Ubicomp '03*, volume 2864 of *Lecture Notes in Comput. Sci.*, pages 149–156. Springer, 2003.

[3] V. Bellotti and W. K. Edwards. Intelligibility and accountability: human considerations in context-aware systems. *Hum.-Comput. Interact.*, 16(2):193–212, 2001.

[4] K. Cheverst, H. E. Byun, D. Fitton, C. Sas, C. Kray, and N. Villar. Exploring issues of user model transparency and proactive behaviour in an office environment control system. *User Modeling and User-Adapted Interaction*, 15(3-4):235–273, 2005.

[5] K. Cheverst, N. Davies, K. Mitchell, and C. Efstratiou. Using context as a crystal ball: Rewards and pitfalls. *Personal Ubiquitous Comput.*, 5(1):8–11, 2001.

[6] A. K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.

[7] A. K. Dey and J. Mankoff. Designing mediation for context-aware applications. *ACM Trans. Comput.-Hum. Interact.*, 12(1):53–80, 2005.

[8] A. K. Dey and A. Newberger. Support for context intelligibility and control. In *Proc. CHI '09*. ACM, 2009.

[9] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *Proc. UbiComp '01*, pages 256–272. Springer-Verlag, 2001.

[10] A. J. Ko and B. A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *Proc. CHI '04*, pages 151–158. ACM, 2004.

[11] A. J. Ko and B. A. Myers. Debugging reinvented: asking and answering why and why not questions about program behavior. In *Proc. ICSE '08*, pages 301–310. ACM, 2008.

[12] B. Y. Lim and A. K. Dey. Assessing demand for intelligibility in context-aware applications. In *Proc. Ubicomp '09*, pages 195–204. ACM, 2009.

[13] B. Y. Lim, A. K. Dey, and D. Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proc. CHI '09*, pages 2119–2128. ACM, 2009.

[14] B. A. Myers, D. A. Weitzman, A. J. Ko, and D. H. Chau. Answering why and why not questions in user interfaces. In *Proc. CHI '06*, pages 397–406. ACM, 2006.

[15] G. Vanderhulst, K. Luyten, and K. Coninx. ReWiRe: Creating interactive pervasive systems that cope with changing environments by rewiring. In *Proc. IE '08*, pages 1–8, 2008.

[16] J. Vermeulen, J. Slenders, K. Luyten, and K. Coninx. I bet you look good on the wall: Making the invisible computer visible. In *Proc. AmI '09*, pages 196–205. Springer-Verlag, 2009.

[17] J. Vermeulen, G. Vanderhulst, K. Luyten, and K. Coninx. Answering why and why not questions in ubiquitous computing. pages 210–213.