

Vision in progress

Rob Tieben

Oct 28, 2009

Product intelligence

A normal product (or system or service), like a chair or drill-machine, should do what the user wants, needs or expects:

a product should do what it is supposed to do, from the user's point of view (1)

If we expand this premise to interactive products, like mobile phones and televisions, we can state that:

an interactive product should react like it is supposed to react, from the user's point of view (2)

Finally, this statement can be extended to include intelligent products; products that behave in a certain way, that adapt to changing conditions, and that somehow can and need to be controlled, like intelligent houses, hovering robots, or auto-dj functions on a mp3 player:

an intelligent product should behave like it is supposed to behave, from the user's point of view (3)

This user's point of view is only formed after the user 'sees' the product's behaviour; the user forms a mental model of what the product should do in different situations. The success of performing those actions determines if the product is intelligent; a lot of mismatches between mental model and actual behaviour make the product (or the user) look less intelligent.

The above implies that for a product to be intelligent, not only the product has to learn, but also the user: the user has to learn how the product behaves, and in turn the product has to learn how to behave in different conditions. Or, in other words, the user *must* have expectations of the product's behaviour, and the product *must* behave in that way.

It does not matter whether this correct behaviour happens through learning algorithms, like envisioned in Philips' Ambient Intelligence vision, or through anthropomorphism that lets the user interpret behaviour as intelligent, like the Pleo dinosaur: intelligence is perceived if the expected and actual behaviour match, in the given conditions and context.

Perceived intelligence in a product is thus determined by two factors:

- (a) the user's prediction of the product's behaviour
- (b) the correct behaviour of the product, as expected by the user

Making a product more intelligent requires the maximising of *both* these factors. We should therefore create a way that increases the user's capability to predict the product's behaviour, while at the same time increases the capability of the product to behave as expected.

End-user programming

One way to optimise the user's prediction of the product's behaviour, and the actual execution by the product, is by letting the user influence or program that behaviour directly. Most of the time, this programming requires a PC-based mode of interaction: the program is created on the PC, uploaded to the product, and the product executes the programmed behaviour. Disadvantages are that the PC-based interaction is totally different from the normal interaction with the product, and that the expected behaviour is difficult to predict based on the separated programming [REFS].

A next step, which has been explored in several projects, is to visualise and allow manipulation of the product's program in the normal use environment itself [REFS]. Or, even more direct, the physical training or demonstration approach: the product is bended or moved, and the program records and repeats this behaviour [REFS]. Yet another approach is the use of tangibles: physical objects that control the product to a certain extent [REFS].

The general problems that occur when making programming more concrete, direct and visible is first of all the coupling of instruction and behaviour: ideally, WYSIWYG functionality should exist in the use context of the product. Second, more complex and abstract programming should also be possible, without losing the concrete programming advantages.

actDresses

In the semiotic theory, it is argued that a sign can have any form, but only becomes a sign when it has a meaning standing for something other than itself. A sign always consists of a signifier (the manifestation of the sign) and a signified (what the sign refers to). If we translate the use of signs to controlling intelligent products, the signifier should thus 'visualise' a change in behaviour (the signified). Fernaeus and Jacobsson (2009) propose actDresses: "a kind of physical markings that can be directly attached to a digital artefact, and that signifies some property, action, or behaviour of that artefact". The two main differences between actDresses and other forms of visual programming are that (a) the sign is shown in the immediate physical context of the object and (b) that actDresses are meant to represent and produce perceivable actions in the artefact. Fernaeus and Jacobsson continue with several examples of actDresses, using comics, fashion and wearables as metaphors.

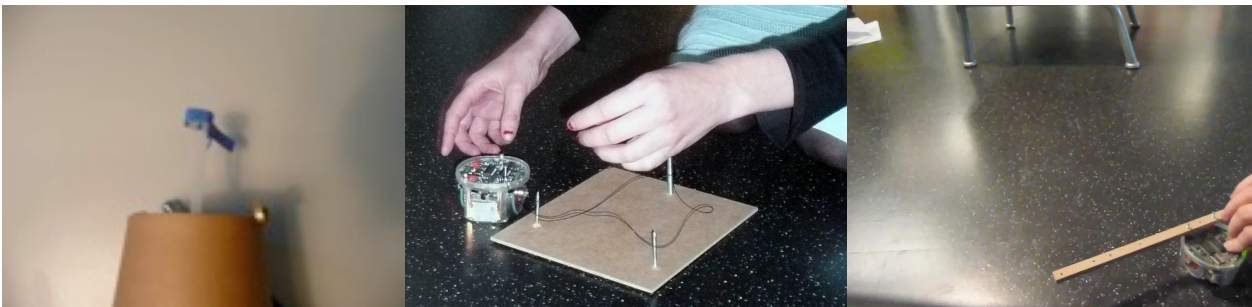
Like in fashion and comics, actDresses are very context dependent: a certain sign in one context could mean something totally different in another context. Basically, actDresses translate contextual signs to programming: each sign indicates what the product will do in this specific context (feedforward), and the product should then execute that action (feedback).

Explorative study

In this study, I want to explore the principle of actDresses: allowing users to program products using physical 'wearables'. Through several iterations of low and mid fidelity prototyping, coupled to fast user evaluations, the actDresses possibilities are explored, with the aim of getting more understanding of the field, and finding and creating a satisfactory solution.

Exploration 1: direct manipulation

In the first exploration iterations, I mainly focused on creating contextual WYSIWYG functionality: can 'robot-wearables' immediately make clear what behavioral change they create, in a certain context? My exploration resulted in direct manipulation: by using signs that physically disable or enable sensors, actuators and the physical characteristics of the robot, the behaviour is also directly changed. Blindfolds for robots (figure 1), strings limiting actuator movement (figure 2) and accessories restricting and guiding movement (figure 3) were created to embody this.



Quick user evaluation and discussions showed that the interpretation is important here, and (too) complex: the user has to understand that disabling an eye sensor will mean that the robot will run into a wall. While this is possible for simple behaviour, and for 'engineer-thinking' users, it will pose a lot more difficulties for complex behaviour: if you physically want to change sensors and actuators to program your robot to pickup a ball and return, you will need a good understanding of all the principles involved.

sign + context + conditions == behaviour
 Δ sign + context + conditions == Δ behaviour
sign + Δ context + conditions == Δ behaviour
sign + context + Δ conditions == Δ behaviour

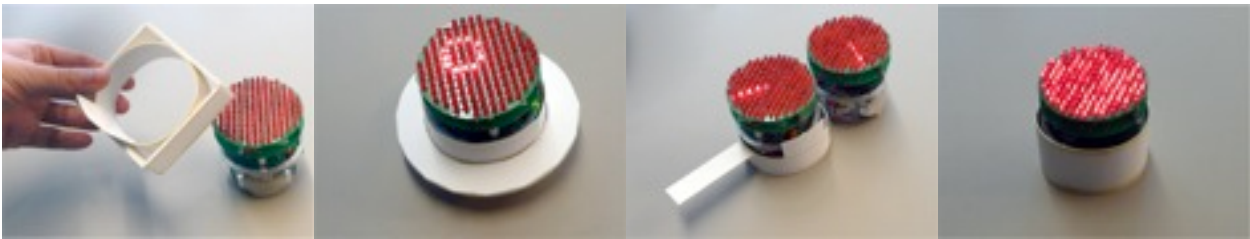
As a second conclusion, this exploration verified the importance of context even more, and added 'conditions' or environment. From the user's point of view, the sign influences behaviour, in a certain context, under certain conditions.

While the sign and the context stay the same, the conditions can and will change, and thus the behaviour will have to change (figure 4).

Exploration 2: direct programming

In the second series of exploration iterations, I started to create the WYSIWYG functionality again: allow users to directly program the robot to behave like they want. As a platform, Glowbots (REF) were used, robots with a LED display on top of them. Creating direct signs that directly manipulate/indicate the robot's behaviour was the goal again.

Eventually, simple shapes were developed that could be placed over the robot: squares, circles, triangles and lines. The robot would then display this shape on the LEDs. When the shape was rotated, the displayed shape would start to rotate as well. Empty robots placed near a shaped robot would copy the behaviour, allowing multiple displays to be programmed. Finally, a 'blocking' shape would reset the robot again (see figure 5, 6, 7 and 8).



The coupling between sign and behaviour is obvious here: programming of different shapes is simple and straight forward. The interpretation of the signs is simple as well; only simple behaviour can be programmed. This exploration already shows the complication and challenge of more complex behaviour: the 'copying' behaviour has no sign at the moment. How could it be made more clear that two shapes copy each other? If this line of reasoning is continued, the real challenge becomes visible: programming complex behaviour using these clear, physical and direct shapes. How would an 'if-then' situation be handled, for example? That is the goal of the next exploration: using this platform of Glowbots to explore signs and complex behaviour.