CHAPTER 18

# Modelling Rich Interaction

## Overview

We operate within an ecology of people, physical artefacts and electronic systems, and this rich ecology has recently become more complex as electronic devices invade the workplace and our day to day lives. We need methods to deal with these rich interactions.

- • Status–event analysis is a semi-formal, easy to apply technique that
  - • classifies phenomena as event or status
  - • embodies naïve psychology
  - • highlights feedback problems in interfaces.

- • Aspects of rich environments can be incorporated into methods such as task analysis
  - • other people
  - • information requirements
  - • triggers for tasks
  - • modelling artefacts
  - • placeholders in task sequences

- • New sensor-based systems do not require explicit interaction, this means
  - • new cognitive and interaction models
  - • new design methods
  - • new system architectures

## 18.1 Introduction

The majority of more detailed models and theories in HCI are focused on the 'normal' situation of a single user interacting with traditional applications using a keyboard and screen. The models focus predominantly on the effects of individual planned user actions. In fact, this 'normal' situation is increasingly looking like the exception. As we noted in the last chapter (section 17.4), much of interaction is about more continuous phenomena both in the computer (e.g. mouse movement) and in more ubiquitous computing environments, such as smart homes, that sense movement, temperature etc. Even traditional computer systems are used not in isolation, but in office and other work settings that involve different people and physical objects. Normal human–computer interaction usually includes looking at pieces of paper, walking around rooms, talking to people. Finally, the more ubiquitous environments deeply challenge the idea of intention behind human–computer interaction: increasingly things simply happen to us.

In this chapter we look at several ways in which this 'normal' model can be modelled either by new methods or by adapting existing ones.

In section 18.2, we will demonstrate how a semi-formal technique, *status–event analysis*, can be used to understand the interplay between more instantaneous events and more continuous status phenomena. The examples in this section will be mainly focused on more traditional computer systems, but unlike the models in previous chapters, status–event analysis is used to describe a ‘slice’ of the system at all levels of abstraction, rather than the whole system at a specific level. Also, by dealing with both more 'computer-ish' events and more 'world-ish' status phenomena it lays the ground for thinking about these increasingly rich interactions as the chapter progresses.

Section 18.3 is still focused on traditional computer systems, but where the emphasis is on the physical and social environment in which they are based. We will be considering how aspects of the rich workplace ecology can be captured in more formal techniques, in particular task analysis. In particular we will look at representing collaboration – who is doing what; information requirements – what do we need to know when; triggers – what makes things happen when they do; artefacts – how to model their behaviour; and placeholders – how we keep track of where we are in a task.

Finally section 18.4 will explode the traditional model completely! The types of systems discussed in ubiquitous computing, and now beginning to be deployed, often do not require explicit interaction. There is a range of levels of intention, from fully intentional systems to those where the system observes and responds to actions of the user performed possibly for some completely different purposes. We will find that these incidental interactions require new ways of thinking about interaction, new ways to design systems and new ways to construct them!

The latter parts of this chapter, especially, involve areas where theory is struggling to keep up with technology and where there is still little idea of where eventually the technology will be used in practice.

## 18.2 Status–event analysis

In Chapter 16 we saw that some dialog notations were state oriented, whereas others were event oriented. Each type of notation had trouble describing some phenomena, but was good with others. Similarly, in Section 17.4 we found that formal models of interactive systems need to be able to deal with both *status* and *event* input and output.

Note that the word ‘status’ is used rather than ‘state’, as the term will be used to refer to any phenomenon with a persistent value. This includes the position of a mouse on a table and the current screen contents, as well as the internal state of the system. The word ‘state’ has connotations of the complete state of the system, rather than the selective particular views meant here by status.

**Note on pronunciation**

Note that the plural of status is not statuses or even stati. Like salmon and sheep, the plural of status is simply status. However, according to the Oxford English Dictionary these are pronounced differently: the ‘u’ in the singular is as the ‘u’ in dat*u*m, whereas the plural has an ‘u’ as in t*u*ne.

The distinction between status and event is between being and doing. Status phenomena always have a value one could consult. For example, you can ask the question ‘what was the position of the mouse on the tabletop at 3:27pm?’ An event, on the other hand, happens at a particular moment. Here the relevant question is ‘at what time did the user press the mouse button?’

This section describes *status–event analysis*, an ‘engineering’-level technique which makes use of the status–event distinction. The label ‘engineering’ is used in a similar way to the way it is applied to the keystroke-level model (Chapter 12, section 12.5.1). An engineering approach is built upon theoretical principles, but does not require a deep theoretical background on the part of the designer. Status–event analysis is built upon two theoretical foundations. On the one side, it is derived from work on formal models of interaction (as described briefly in Section 17.3.4). However, a designer using the method does not need to use, or even know about, these foundations. On the other side, status–event analysis makes use of fairly naïve psychological knowledge, to predict how particular interface features affect the user.

The strength of the method is that a single descriptive framework can be applied at a range of levels from the application, through the interface, to the user’s perception. Indeed, the same descriptive framework can describe even the low-level electrical signals and logic in the microseconds from when a user hits a key to that key being ‘noticed’ by the system.

We will first consider an example of clocks and calendars, which demonstrates some of the important properties of events and status and how they interrelate. The design implications of this are discussed in Section 17.4.2. In particular, we will see that events generated by applications have an associated time-scale which tells us when we *want* them to be perceived by the user. Section 17.4.3 discusses a few simple psychological facts which help us to predict when interface events become salient for the user.

Event/status analysis looks at different layers of the system, such as user, screen (presentation), dialog and application. It looks for the events perceived at each level and the status changes at each level. This, combined with the naïve psychological analysis of the presentation/user boundary, allows the designer to predict failures and more importantly suggest improvements. This approach is demonstrated in two examples: the ‘mail has arrived’ interface to an email system and the behaviour of an on-screen button.

## 18.2.1 Properties of events: clocks and calendars

Brian is due to meet Alison to go to the cinema at 20 to 8. He decides he will stop work at 25 to, and keeps an eye on his watch. Every few minutes he looks at it, increasingly frequently as the time draws nigh. Eventually, he looks and it is 24 minutes to, so he quickly puts his coat on and leaves.

In fact Brian had an alarm on his watch. He could have set it for 7:35, and waited for it to ring. Unfortunately, he has never worked out how to set the alarm (nor how to stop it beeping every hour).

A few days later Alison is sitting in her office. In an idle moment she consults her calendar to see what is happening tomorrow. She sees that it is Brian’s birthday, so decides to buy him the soundtrack of the film they recently saw.

From these scenarios, we can abstract many of the important properties of status and events:

**Status** Brian’s watch is a status – it always tells the time – so is Alison’s calendar. Moreover, assuming Brian’s watch is analog, this demonstrates that status phenomena may be both discrete (the calendar) or continuous (the watch face).

**Events** The passing of the time 7:35, when Brian wanted to stop work, was an event. A different, but related, event was when Brian got up to go. The alarm on Brian's watch (if he could use it) would have caused an event, showing that Brian's watch is capable of both status and event outputs. Alison also experienced an event when she noticed it was Brian's birthday the next day, and of course, his birthday will also be an event.

**Polling** Given Brian only had a status – the watch face – and he wanted an event – 7:35 – he looked periodically at his watch. In computing terms, Brian *polled* his watch. Polling is a normal activity that *people* do as well as machines. It is a standard way to turn a status into an event.

**Actual vs. perceived** The event Brian was after was when the watch said 7:35. This event happened, but Brian obviously did not look at his watch at just the right moment. Instead, this *actual event* became a *perceived event* for Brian a minute later when he next looked at his watch. If one looks at a fine enough time-scale there are almost always gaps between actual and perceived events. Of course, there can be similar lags between actual and perceived status too.

**Granularity** The watch showing 7:35 and Brian's birthday are both events, but they operate at completely different time-scales. The interpretation of events and status may differ depending on the time-scale one uses. In particular, the idea of immediacy changes.

These same properties all emerge during the analysis of interactive systems.

### 18.2.2 Design implications

Applications want to cause events for users and use various presentation techniques to do this. However, these techniques must be matched to the *time-scale* of the desired event. For example, if a stock of 6 mm bolts is running low, this requires reordering within days or weeks. On the other hand, a coolant failure in a nuclear power plant may require action within seconds.

The presented form of the event for the user must match these time-scales – both causing events too fast or too slow is wrong. It is fairly obvious that too slow an event is wrong. An email message to the power plant operator would be ineffectual; the operator and the computer would both be so much radioactive waste. However, the opposite fault can be equally damaging. Red flashing lights and alarm bells when the last box of 6 mm bolts is opened would be annoying, and could also distract the operator from more important tasks, such as dealing with that coolant.

A less extreme example would be an electronic alarm and calendar. Imagine we have an on-line alarm function, which can be set to sound a buzzer and put a message in the middle of the screen at any time we like. This would obviously have been useful for Brian who could have set it to say 'cinema with Alison' at 7:35. However, if Alison wanted to remind herself of Brian's birthday, she would be forced to set an alarm for a specific time, say noon on the day before. This would have been disruptive when it rang, and not in keeping with the time-scale of birthdays.

In order to cause a perceived event for the user at the appropriate time-scale, we must be able to predict the event time-scale of various interface techniques. Simply presenting information on the screen, or causing an event at the interface, is no guarantee that that event will become a perceived event for the user.

### *18.2.3 Naïve psychology*

In order to predict the effect of interface techniques, we need to employ some naïve psychology. This can tell us what sort of stimuli are salient and where the user's attention will be focused.

First, we can sometimes predict *where* the user will be looking:

**Mouse** When the user is positioning the mouse pointer over a target, the user's attention will be focused on that target. This is guaranteed in all but a few situations by the feedback requirements of hand–eye coordination. However, this attention may not stay long after the target has been successfully 'hit'.

**Text insertion point** While typing text, the user will intermittently look at the text just typed and hence the current insertion point. However, because of touch-typing, this is less certain than the mouse except when moving the insertion point over large distances using cursor keys – another positioning task.

**Screen** It is reasonably safe to assume that the user will look at the screen intermittently. However, there is no guarantee that any particular message or icon on the screen will be noticed, only that very large messages spread across a large part of the screen will probably be noticed.

If we know where the user is looking, then we can put information there (not in a status line at the top where no one ever looks). Also, changes at the user's visual focus will be salient and become a perceived event for the user. An example, where the mouse pointer itself is used for information, is the egg-timer or ticking watch icon used when a system is busy.

Secondly, immediate events can be caused even when we do not know where the user is looking. The most common are *audible events*: beeps, buzzers, bells and whistles. These cause perceived events even when the user is not looking at the screen. In addition, our *peripheral vision* is good at detecting *movement* (see Chapter 1). Whereas we might not notice a small *change* unless it is in our visual focus, we will notice something moving out of the corner of our eye. We will see an interesting example of this in the next section, but a common example of *large* change (rather than movement) is the use of a screen flash as a silent bell. Not only does this cause an event when you are looking anywhere on the screen, but even if you are looking at the keyboard or at a document beside the screen. The only proviso is that the duration of the flash must be timed suitably to avoid it being mistaken for normal screen flicker.

Finally, recall from Chapter 1 that when people complete some goal, they experience *closure*. This means that they have a feeling of completeness and go on to the next thing. Closure has implications both on perception and actions. It is why in the mouse positioning task, the user's eye may stray from the target as soon as the target is perceived as 'hit'. In addition, the user may begin some of the actions for the next task, while certain automatic actions terminating the last task are still going on. For example, it is easy to knock a glass from the table by beginning to turn round before fully letting go of the glass.

We will see examples of each of these three effects in the succeeding subsections.

### *18.2.4 Example – email interface*

Brian wants to thank Alison for his birthday present, which she left on his desk. He sends her a message by email. Consider the stages the message goes through, from when the message first arrives in Alison's system until Alison realizes it is there.

To make life easy we will assume that Brian is on the same network as Alison. When he hits the ‘SEND’ button on his machine, the message is sent – this is an event. The way that many systems handle internal mail is simply to append it to the recipient’s mailbox file, for example ‘/usr/spool/mail/Alison’. The *event* of receiving mail is therefore reflected in a change of *status* in the file system. You can see this event depicted as the first arrow in Figure 18.1. This figure shows timelines for various components in the interface, where time flows downwards. Events are denoted by arrows between the components’ timelines.

[[[ **** **Old 9.5 p369 **** ]]]**

Figure 18.1 Inputs and outputs of single-user system

On Alison’s workstation runs a mailtool. When not in use, the mailtool is depicted by an ‘empty mailbox’ icon. The mailtool does not notice the change in the mailbox file immediately, but periodically it checks the file to see if it has changed – that is, it *polls*. So, after a while, the mailtool polls the file and sees that it has indeed changed. At this point the change in *status* of the file system has become a *perceived event* for the mailtool. Notice that we are using the term perceived event of computer agents as well as of the user. Obviously, the final perceived event for the user is what is important, but we are also interested in similar phenomena at different levels.

Having noticed the event, the mailtool now knows that mail has arrived, and must try to make this event a perceived event for the user. To do this it changes its icon to denote a mailbox with a letter sticking out. That is, we again see an *event* giving rise to a change in *status*, this time on the screen.

Finally, we come to the user, Alison. She is sitting at her workstation busy on a report she must finish. She gets to the end of a difficult section and breaks in her typing for a moment. During such breaks, her eyes wander over the screen, and in particular she occasionally glances at the mailtool’s icon to see if any mail has arrived – she *polls* it. This time when she looks up, she sees that mail has indeed arrived – the mail arrival has at last become a *perceived event* for Alison.

If we look at Figure 18.1, we see that a number of active agents (Brian, the mailtool and Alison) cause events for one another mediated by status elements (the filestore and screen). This is a very common scenario, especially if you look at fine details of interaction. However, it is also possible to have direct event-based connections (which we will see in the next example), or even status–status connections. An example of the latter is the linkage between the mouse on the table and the mouse pointer on the screen. Even this is mediated by events in its implementation, but this is not apparent to the user.

Having analyzed the event/status dynamics of the system, we can ask whether it is functioning as it should. In fact, for this particular message and context it functioned well enough, but let us consider a few alternative scenarios.

If the message had been ‘Fire! Get out quick’ Alison might now be dead. Forgetting the interface design, the mailtool probably does not poll the file system often enough to respond within the time-scale of such a message. If the system were required to support messages of such urgency, we would need to redesign the mail arrival mechanism so that the mailtool would receive a direct event, rather than wait to poll. Assuming this were done, would it have saved Alison? Probably not, because she would not have looked at the mailtool icon sufficiently often to see the crucial message.

On the other hand, the message may have been information about a forthcoming conference. Alison need not have read this message when she did. The perceived event is now at too fine a time-scale, and it is an unwanted interruption.

Finally, if at 12:30 Brian sent the message ‘Thanks for the gift, see you for lunch at 1 o’clock?’, then the time-scale may be appropriate, but the guarantee of delivery of the current system is too weak. Alison usually glances at the icon every few minutes, but occasionally, when engrossed in a task, she may miss it for hours. Alison has at least survived, but is getting hungry.

Split-second requests are not normally sent by email and so the last form of message is the most urgent encountered in typical email traffic. The time-scale required is of the order of a few minutes. But we saw that the current interface, although of the appropriate time-scale, does not carry sufficient guarantees. There are other interfaces available, so we shall see how they fare:

**Explicit examination** The traditional email interface required the user to examine the mailbox explicitly, say in the morning and evening. This was a form of polling, but at a much reduced time-scale. This would obviously be useless for Brian’s message, but would have been much more appropriate for the conference announcement.

**Audible bell** The existing mailtool can be set to sound a bell when mail arrives. This would cause an instant perceived event for Alison – if she was there. To avoid being missed entirely when Alison is out of the room, the bell has to be combined with a *status* indicator, such as the icon. However, even if Alison were there, the interruption caused to her work would not merit the normal time-scales of email messages – unless it said ‘Fire!’, that is.

**Moving faces** Finally, there is a second mail-watcher available, which when mail arrives sees who it is from and slowly moves a bitmap picture of the sender into a sort of ‘hall of fame’ at the bottom of the screen. Whereas normally the mailtool icon is not noticed as it *suddenly* changes, this *movement* is noticed at once as it is in Alison’s peripheral vision. Furthermore, it leaves a status indicator behind (the sender’s face). It thus does the job of the buzzer and icon combined. However, the guaranteed event is still too quick.

What is really wanted is a guaranteed event at a time-scale of minutes. None of the available options supplies this. However, knowing what is wanted one can suggest designs to supply the need. For example, we could automatically notice gaps in typing, and notify the user (aurally or visually) during a gap on the assumption that this will be less obtrusive. Alternatively, we can use a non-guaranteed technique of the appropriate time-scale, such as the existing mailtool icon, but if the mail is not examined within a certain time we can use a more salient alarm.

Ideally, such mechanisms should be tuned to the particular time-scale of the application and, if anything, email is one of the most difficult examples as the time-scale depends on the message. Other applications, particularly command and control tasks, will have more well-defined time-scales making the matching job easier.

### *18.2.5 Example – screen button feedback*

The last example used status–event analysis to suggest an improved interface to email. However, email is, as we admitted, a complex example, so it is not surprising that improvements can be found. In the following example, we find that status–event analysis is even able to suggest improvements in something as simple and heavily used as an on-screen button.

Screen buttons activated by clicking the mouse over them are a standard *widget* in any interface toolkit and are found in most modern application interfaces. The application developer has little control over the detailed user interaction as this is fixed by the toolkit. So, the specific results of this example are most relevant to the toolkit designer, but the general techniques are more widely applicable.

A common problem with many on-screen buttons is that the user thinks the button has been pressed, but in fact it has not been. As an example, imagine Alison at work again on her word processor. The report is too long and so when she notices a superfluous paragraph, she selects it and then moves her button up to the 'delete' button. She clicks over the button and thinks it has had an effect, but actually as she lifted her finger from the button, the mouse slipped from the button and the click was ignored (the button is activated by the mouse *up* event). Unfortunately, she does not notice until having, with difficulty, pared the report down to 1000 words, she notices that the unwanted paragraph remains.

We have two questions: why is this mistake so frequent, and why didn't she notice? To answer these we use status–event analysis to look at two scenarios, the first where she successfully selects 'delete', and the one where she does not. There are four elements in the analysis: the application (word processor), the button's dialog (in the toolkit), the screen image and the user (Alison). Figures 18.2 and 18.3 depict the two scenarios, the first when successful – a hit – and the second when not – a miss.

Consider first the successful case in Figure 18.2, the hit. The first significant event is Alison's depression of the mouse button over the on-screen 'delete' button. This event goes directly to the toolkit dialog, which responds by highlighting the 'delete' button. The next event is as Alison lifts her finger from the button. Again this is received by the dialog which this time does two things: it removes the highlight from the 'delete' button, and also causes an event 'delete' for the application. The application then performs the action, deleting the paragraph. The effects of this change in the text are reflected in the screen content.

[[[ **** **Old 9.6 p372 **** ]]]**

Figure 18.2 Screen button – hit

The unsuccessful case (Figure 18.3, the miss) starts similarly. Alison depresses the mouse button and receives feedback. However, this time, before releasing the mouse button, she accidentally moves the mouse off the button. The toolkit dialog responds to this by removing the highlight from 'delete' – the same feedback as in the first scenario. Alison's release of the mouse button has no further effect.

[[[ **** **Old 9.7 p372 **** ]]]**

Figure 18.3 Screen button – miss

The two scenarios are very different in their effect: in one the application deletes some text, in the other it does not. However, Alison does not notice the difference. Her feedback from the toolkit dialog is *identical*. In theory, she could have seen that the text did not change as she expected. However, after hitting the 'delete' button, she reaches *closure* on that operation and moves on to the next task. Her attention is not focused on the text to be deleted and so there is *no perceived event* for the user corresponding to the application event of the text being deleted.

Furthermore, this closure makes the mistake not just a possibility, but highly likely. Consider the moment when Alison has just pressed down the mouse button and the on-screen 'delete' button has been highlighted. She has done what she wanted and attains closure, and the remaining release of the mouse button is initiated. She now starts to look for the next action and begins to move the mouse to the location of next interaction. However, the two actions, releasing the mouse and moving it, are not synchronized with one another. There is no particular reason why one should happen before the other. It is, of course, a particularly dangerous point in a dialog where the order of two unsynchronized user actions makes a crucial difference to behaviour.

It is quite difficult to see how to avoid the problem occurring. It is not that the current feedback is not salient; it is at the focus of the pointing task. However, all the feedback concerns events at the dialog level. The most important event, the ‘delete’ to the application, has *no* corresponding perceived event. The toolkit assumes that the user will see some feedback from the application and therefore does not supply any feedback of its own. But, as we saw, the application’s feedback is very likely to be missed.

The solution is fairly obvious: the dialog should itself supply an event, which will be perceived by the user, corresponding to the application-level event. This could be visual, but would have to be very salient as the user’s eyes are beginning to move towards the next task. Alternatively, it could be aural, either with a keyboard-like ‘click’ as the button is successfully pressed, or with a beep if the mouse slips off. This improved feedback could be combined with some dynamic mechanism, such as making the screen button ‘magnetic’ and difficult to move out of.

It is interesting to note that, if Alison were a novice user, she would be more likely to check her actions and thus notice the mistake – an unnoticed button miss is an *expert slip*. As all but the most extensive user testing of a new device must, by definition, be with novices, there is no way this would be detected – which is perhaps why most on-screen buttons have this problem. We hope this demonstrates how, on occasions, semi-formal hand analysis may even be more effective than real user testing.

## 18.3 Rich Contexts

Formalised methods such as task analysis adopt a systemised, almost Taylorist view of the work place – people working to achieve well-defined goals following regular procedures.

However, even the earliest systems analysis texts did take into account the richness of the work environment. One text, written in the late 1960s, described a printshop where productivity was lower than predicted after the installation of new machinery. The analyst was asked to advise on automating the equipment. After observing the workplace he asked for a small budget of a few hundred pounds and the productivity dramatically rose. What did he do? He bought white overalls. The equipment was oily and the operators, mostly young women, were reluctant to work too quickly for fear of damaging their own clothes. The overalls protected their clothes and obviated the need for a computer.

This is not a unique story. Again and again those studying real workplaces find that they have a rich ecology involving different people, the structure of the spaces they work in and the physical artefacts in the workplace. Observations of real photocopier use led to the ideas of situated action (see also Chapter 13, section 13.3.4) challenging simplistic models of pre-planned human action and proposing instead that real interaction is not pre-planned, but rather acted out in response to the actual work situation [[S87]]. Numerous ethnographic studies emphasise the incredible richness of human interaction and, often, the inability of formalised processes to incorporate it. For example, in a study of a printshop (yes, another) Bowers et al. [[BBS95]] found that the operators constantly had to work around the job management software as it assumed linear patterns of work that did not reflect the contingent and dynamic re-planning necessary on the shopfloor.

In a philosophically different strand of work, the *distributed cognition* literature has challenged the model of cognition “in the head” and instead suggests that real cognition happens in interaction with the environment and with each other (see Chapter 14, section 14.5.3). One classic study showed how Polynesian sailors were able to navigate without formal charts and without the requisite experience in any one individual’s head [[H90]].

One could say that the lessons of situated action and of distributed cognition are about the parity in relationship between the ‘actor’ and the world. We do not just *act on* the world, but *act with* the world. We are driven by what we see and hear from other people, from automated systems and from the physical objects in the world. In response our actions, words and sometimes gestures and demeanour speak back into that rich world.

In day to day life we understand about dialogue with other people. In HCI we are used to thinking about dialogue between users and the computer system. However, in a full ecological analysis we must also accept that users are in dialogue with the physical environment. We use the information stored in artefacts and their physical disposition to trigger and guide our actions, and the physical properties of the world limit and constrain our actions on it.

In the rest of this section we will look at several phenomena of this dialogue with the environment and see how they can be grafted into more traditional methods.

### 18.3.1 Collaboration – doing it together

In Chapter 14, we discussed issues of communication and collaboration. However, you may notice that this is rarely mentioned in the other models in Part 3.

In fact several notations and methods do handle collaboration explicitly. There are two ways in which this can be done. One is where the process as a whole is mapped out and parts assigned to each person (common in function allocation, workflow and process methods). The other is where several role-oriented models interact. These are complementary representations and can be handled together with suitable tool support.

One example is *ConcurTaskTrees* (CTT) which are a form of hierarchical task analysis [[P99]]. CTT adds to HTA in two main ways. The first is that instead of the loosely described plans of HTA it includes a much more formal way of specifying the temporal relationships between subtasks using operators based on the LOTOS formal notation. A CTT task tree can be produced for each person involved in the task. It is, however, the second difference from HTA that is significant here. Where several people collaborate on a task, a larger task tree is produced where each subtask can be labelled as belonging to a specific individual, being automated, or being collaborative (see figure 18.4).

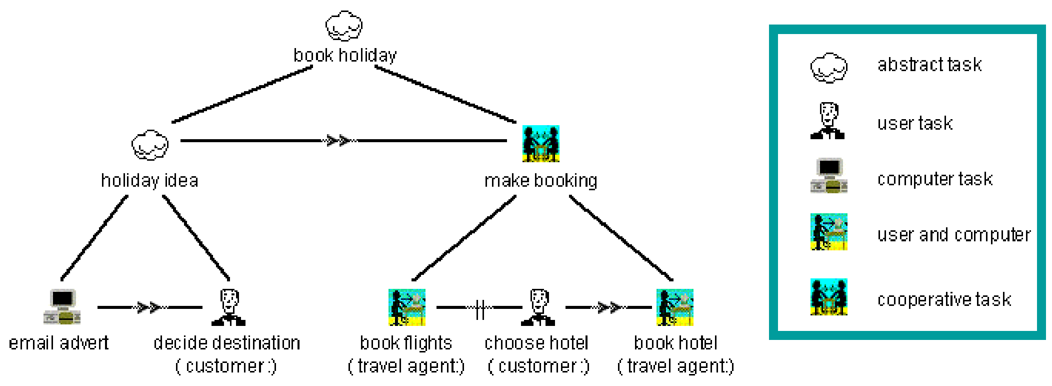


Figure 18.4 ConcurTaskTree [[P99]]

It is interesting to note that the roles identified in CTT include both humans and automated systems, but not aspects of the physical environment. However, it is only a small step to imagine treating the environment or parts of it as dialogue partners alongside the human and computer.

Another method that takes collaboration seriously is *GroupWare Task Analysis* (GTA), which includes a broad-ranging conceptual framework, elicitation techniques and toolset [[VW00]]. GTA has a rich taxonomy including agents and roles for modelling collaboration and objects both physical and electronic (see figure 18.5). It aims to build a rich model of the situation in which tasks are performed.

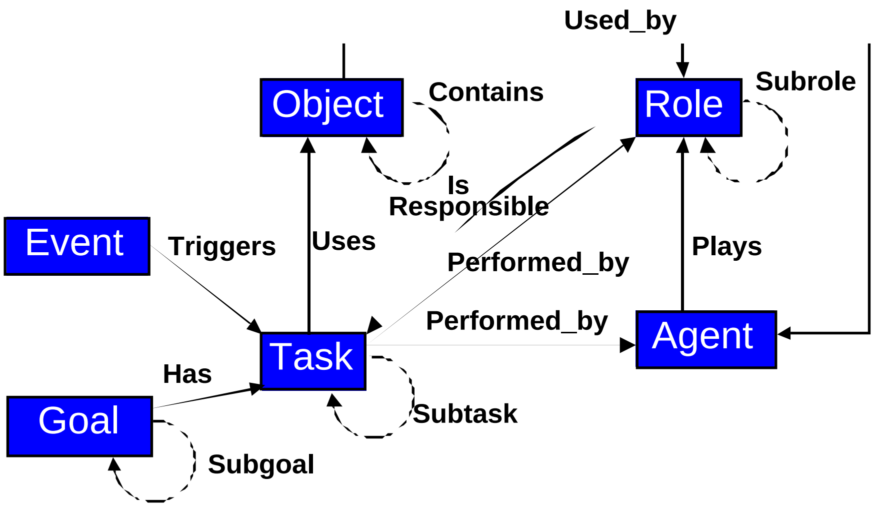


Figure 18.5 GTA ontology

### 18.3.2 Information – what you need to know and when you need to know it

When writing the first edition of this book, we bemoaned the fact that cognitive models took a view of human cognition that was almost totally dominated by output and action (see section 12.4). We have goals, which translate into sub-goals, and so on, until we perform actions – an entirely head-outward flow of control. In a similar vein, Lucy Suchman’s theories of *situated action* were particularly critical of the AI inspired views of human planning. These models of planning are largely based on creating internal plans based on internal models of the world, which are then ‘blindly’ executed. We use the word ‘blindly’ here quite carefully, as these are models of human action which ignore the human senses entirely.

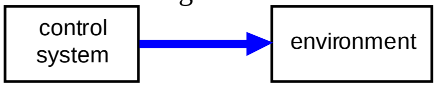


Figure 18.6 open loop control

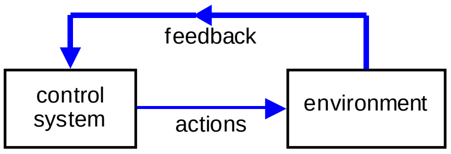


Figure 18.7 closed loop control

In control engineering these output-only models would be described as open-loop control (figure 18.6) as opposed to closed-loop control (figure 18.7), which constantly monitors the effects of its outputs on the environment and uses these to modify future behaviour.

In general, closed-loop control is more robust and it is not surprising that both internal physiological processes and external human behaviour are typically closed-loop systems. Indeed, the user interface literature is full of the importance of feedback and effective information display; it is just that the early formal models have often left this out.

There are several examples of cognitive models that do take this feedback loop seriously. As noted in Chapter 12 (section 12.4), there is a display-based version of task action grammar [[HP90]], and there have been several other variants of display-based models. Also, interacting cognitive subsystems (ICS) (section 12.6.2) is focused strongly on the transformations of representation during the perception-to-action cognitive cycle [[BM95]]. The earliest papers on cognitive complexity theory (CCT) included perceptual operators on the production-rule-based cognitive model component, but strangely it was the actions only that were matched against the system dialogue model (section 12.2.2).

It is not uncommon to see references to information seeking in the names of tasks in task models, but this is normally where the information seeking activity is regarded as a substantive task. In practice, information is used throughout task execution. For example, in the simple tea making task (figure 18.8), the “boil kettle” subtask does not require any information, but the “get out cups” task requires the actor to know how many are required. Does he remember, or does it need to be written down?

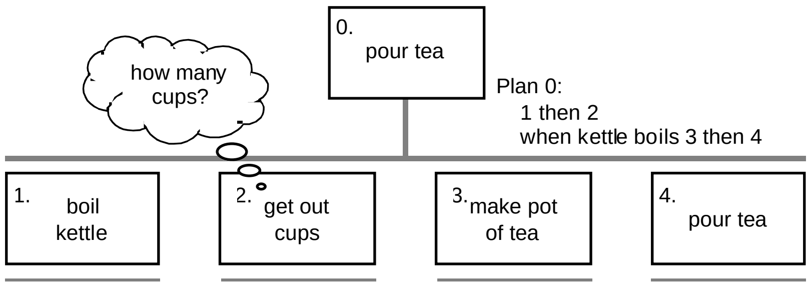


Figure 18.8 tea making task

Information is central to several task analysis methods, such as TAKD (Chapter 15, section 15.4); however, these are focused on what kinds of things the user needs to know in general – ontology and domain modelling – not on what the user needs to know at a particular moment.

It is a simple matter to add an information analysis stage to any task analysis method or notation. Note that some tasks have no information requirements – other than the fact that they are to happen. For example, the “make pot of tea” subtask requires no information other than the fact that the kettle has boiled. However, information is required whenever:

(a) a sub task involves inputting (or outputting) information

(b) there is some kind of choice

(c) a subtask is repeated a number of times that is not prespecified

Note that (c) is a special case of (b). To detect (a) one needs to look at the kind of task, whereas (b) and (c) are evident from the temporal structure of the task (for example, in the case of HTA, this would be in the plan).

Having discovered that information is required it may come from several sources:

(i) It is part of the task (e.g., in the case of a phone call, whom one is going to phone)

(ii) The user remembers it (e.g. remembering the number after ringing directory enquiries)

(iii) It is on a computer/device display (e.g. using a PDA address book and then dialling the number)

(iv) It is in the environment: either pre-existing (e.g. number in phone directory), or created as part of the task (e.g. number written on piece of paper)

Reducing memory load is part of standard usability guidelines. Knowing what information is required during a task allows us to design or redesign the task so that information is available when required. An infamous example of this is those all too common modal dialogue boxes that ask you some question but hide the window containing the information you need to answer the question!

In most multi-windowed GUIs it has been possible for user interface designers to be quite careless about information requirements. One can make so much information available and let the user arrange different windows to perform the task. In contrast industrial control design is far more careful about knowing what is required, as there are often very many possible values to display. Industrial operators may have very little time to respond to an alarm and so cannot browse complex menu systems to find information. As user interaction moves away from the computer screen to dedicated devices, WAP phones, interactive television screens and smart appliances, these issues of careful information requirements analysis will become significant for all applications.

### *18.3.3 Triggers – why things happen when they happen*

Workflows and process diagrams decompose processes into smaller activities and then give the order between them. Similarly, plans in HTA give some specification of the order of subtasks and, as noted earlier, in CTT these temporal orders are made more specific using operators derived from LOTOS.

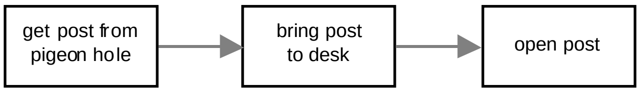


Figure 18.9 simple work process

Figure 18.9 shows a simple example, perhaps the normal pattern of activity for an office worker dealing with daily post. Notice the simple dependency that the post must be collected from the pigeonhole before it can be brought to the desk and before it can be opened. However, look again at the activity "open post" – when does it actually happen? The work process says it doesn't happen before the "bring post to desk" activity is complete, but does it happen straight away after this or some time later?

*Trigger analysis* [[DRW03]] looks in detail at the triggers that cause activities to happen when they happen. In the case of opening post this could easily be something like "at coffee time" rather than straight away. It identifies a number of common triggers:

- *immediate*: straight after previous task
- *temporal*: at a particular time or after a particular delay
- *sporadic*: when someone thinks about it
- *external event*: some event occurs, such as a phone call
- *environmental cue*: something in the environment prompts action

We can augment the work process with triggers for each activity (figure 18.10).

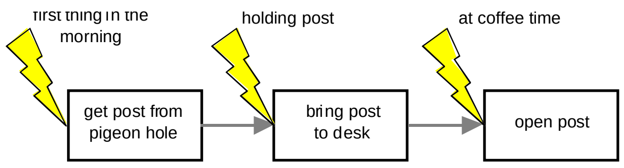


Figure 18.10 triggers for activities

Notice how we have examples of different types of trigger: two temporal and one environmental (letters in the office worker's hand prompting her to carry them to her desk).

Triggers are important not only for understanding the temporal behaviour of the task, but also because they tell us about potential failure modes. If two environmental triggers are similar, one might do parts of the task out of sequence; if a trigger may not occur, or may be missed (likely for sporadic triggers), activities may be omitted entirely. Triggers also help us assess the likelihood of problems caused by interruptions – for example, immediate "just after" sequences are disrupted badly, whereas environmental cues tend to be robust (because they are still there).

Sometimes triggers are seen in the plans of HTAs, and sometimes 'waiting' subtasks are included for external events, but these are both exceptions; the normal assumption is that tasks are uninterrupted. However, it is straightforward to add a trigger analysis stage to most task analysis methods.

In addition, you may have noticed that the ontology of GTA in figure 18.5 includes events and triggers. However, the word 'trigger' in GTA is usually used only for events that originally set a task in motion (e.g. a customer making an order) and events that make major changes (e.g. the customer ringing to cancel the order).

In terms of the ecology of interaction, triggers remind us that tasks are not typically performed uninterrupted and continuously from start to finish. In practice, tasks are interleaved with other unrelated tasks or, potentially more confusingly, with different instances of the same tasks, and may be interrupted and disrupted by other activities and events. Furthermore, the performance of the tasks is dependent both on a host of interactions with the environment – and these may be fragile – and on apparently unconnected events.

### *18.3.4 Artefacts – things we act on and act with*

Notice that one of the trigger types is environmental cues – things in the environment that prompt us to action. Some years ago Alan got a telephone call reminding him to respond to a letter. He couldn’t recall receiving it at all, but searching through a pile on his desk he found it, and several other letters from a period of several weeks unopened and unread. What had happened? His practice was to bring the post upstairs to his desk, but not always to read it straightaway. Not being a coffee drinker, it was not coffee time that prompted him to open the post but just the fact that there was unopened post lying on his desk. This process had worked perfectly well until there was a new office cleaner. The new cleaner didn’t move things around on his desk, but did ‘tidy’: straightening up higgledy-piggledy piles of paper. However, he had unconsciously been using the fact that the bundle of unopened post was not straight as a reminder that it needed dealing with. So post that for some reason was not opened one day would look the next morning as if it was tidily ‘filed’ in a pile on his desk.

This story is not unique. The ethnographic literature is full of accounts of artefacts being used to manage personal work and mediating collaborative work. Some of that purpose is to do with the content of the artefacts – what is written on the paper – but much of it uses the physical disposition: by orienting a piece of paper towards a colleague you say ‘please read it’. In the case of Alan's desk, the cue that said “post needs to be opened” was purely in the physical orientation (not even the position).

Of course, artefacts also carry information, and are often the inputs or products of intellectual work. Furthermore, in physical processes the transformation of artefacts is the purpose of work.

One example that has been studied in detail in the ethnographic literature is air traffic control, in which all these uses of artefacts are apparent [[HORSR95]]. Flight strips are central (figure 18.11) – small slips of card for each aircraft recording information about the aircraft (flight number, current height, heading, etc.). This information is important both for the controller managing the aircraft, and also as an at-a-glance representation of the state of the airspace for other controllers. However, the controllers also slightly pull out strips corresponding to aircraft that have some issue or problem. This acts partly as a reminder and partly as an implicit communication with nearby controllers. Finally, the strips in some way represent the aircraft for the controllers, but of course the real purpose of the process is the movement of the aircraft themselves.

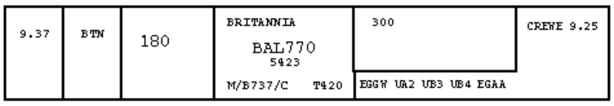


Figure 18.11 air traffic control flight strip

Task models often talk about objects, either implicitly in the description of subtasks or explicitly in the task model. However, the objects are always ‘second class’ – users act on them, but they are not ‘part of’ the task. CTT and most work process notations do talk about automated tasks, but not about the artefacts, whether electronic or physical, included within the interaction.

In object-oriented design methods it is common to give lifecycle descriptions of ‘objects’; however, this is usually because we are intending to store and automate the object electronically. And though workflow analysts do study document lifecycles, this again is largely because of the intention to automate.

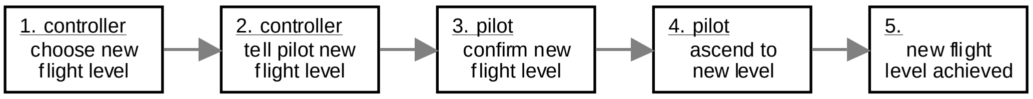


Figure 18.12 flight level management task

The entity–relationship style task analysis in Chapter 15 (section 15.5), based largely on the ATOM method [[W89]], does treat physical objects as 'first class', but this type of method has not gained widespread acceptance.

There is no reason why most task analysis methods should not adopt some form of artefact tracking. This may be as simple as recording which artefacts are triggers for, used by, modified by, or produced by any particular subtask. For tasks where artefacts are particularly central, more sophisticated artefact lifecycles could sit alongside the task description. These lifecycles may be mundane (letter closed – letter open), but this is the point; users recruit their everyday knowledge and physical properties of the world to coordinate their activity.

### *18.3.5 Placeholders – knowing what happens next*

It is half past five in the evening. The busy office building is beginning to quiet as people pack up to go home. One or two work late in their offices, but as the evening wears on they too go home. Soon there is only the hum of vacuum cleaners and the clatter of wastebins as the office cleaners do their work, until eventually, the last light goes out and the building sleeps. A few have taken papers and laptops home and continue to work, but eventually they too put aside their work and sleep.

It is three o'clock in the morning. In the darkness and silence of the office and the deep sleep of all the employees, where is the memory of the organisation? The next morning at nine o'clock the office is a flurry of activity; it has not forgotten and has restarted its activities, but how?

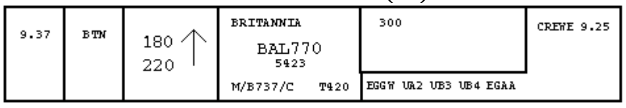
We have already discussed two aspects of this memory: information required to perform tasks, and triggers that remind us that something needs to happen. However, there is one last piece of this puzzle that we have hinted at several times already. As well as knowing *that* we need to do something, we need to know *what* to do next. In the complex web of tasks and subtasks that comprise our job – *where* are we?

In fact, in looking at triggers we have already seen examples of this. The untidy post on Alan's desk said "something needs to happen", but the fact that it was also unopened said, "it needs to be opened". We already noted that similar triggers may cause subtasks to be performed out of sequence. If we have only a small number of dissimilar tasks this is unlikely to happen, since we can remember where we are in each task. However, as the number of tasks increases, especially if we are performing the same task on different things, it becomes harder to remember where we are.

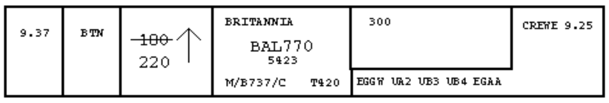
Let's look again at air traffic control. One of the controller's tasks is to manage the flight level of aircraft. A much-simplified model of this activity is shown in figure 18.12. Because this is a shared task between the controller and the pilot, each box is labelled with the main actor (although tasks 2 and 3 are both communications). Recalling earlier parts of this section, we might ask what information is required at each stage; for example, task 1 would depend on radar, locations of other planes, planned take-off and landings, new planes expected to enter airspace.

Note that box 5 is not really a task, more a 'state of the world' that signifies task completion; however, it is important, as the controller will need to take alternative actions if it doesn't happen. Of course, without appropriate placeholders the controller might forget that a plane has not achieved its target level, causing problems later because the old level is still occupied and allowing potential conflicts between aircraft.

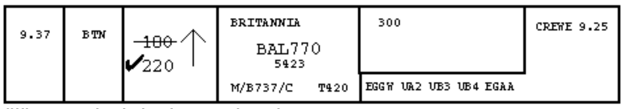
In fact, the flight strips do encode just such a placeholder (see figure 18.13). When the controller informs the pilot of the new height he writes the new level on the flight strip (i). When the pilot confirms she has understood the request the controller crosses out the old level (ii). Finally when the new level has actually been reached the new level is ticked (iii).



(i) controller gives instruction to pilot "ascend to flight level 220"



(ii) pilot acknowledges the instruction



(iii) new height is attained

Figure 18.13 flight strip annotated during task

Virtually all task-modelling notations treat the placeholder as implicit. The sequence of actions is recorded, but not why the user should do things in the way proposed. Of course, one purpose of task analysis has been to produce training – that is to help people learn what the appropriate processes are, but this doesn't help them to actually remember where they are in the process.

Just like other forms of information, placeholders may be stored in different ways:

(a) in peoples' heads – remembering what to do next

(b) explicitly in the environment – to-do-lists, planning charts, flight-strips, workflow systems

(c) implicitly in the environment – is the letter open yet?

Although often forgotten, placeholders are crucial in ensuring that tasks are carried out effectively and in full. At a fine scale it is rare to find explicit records because the overhead would be too high. Instead (a) and (c) predominate. As users' memories may be unreliable when faced with multiple tasks and interruptions, it is not surprising to find that various forms of environmental cue are common in the workplace. However, electronic environments do not have the same affordances to allow informal annotations or fine 'tweaking' of the disposition of artefacts.

## 18.4. Low intention and sensor-based interaction

In traditional computer applications a user was expected to approach the system with a clear intention to perform some activity or achieve some goal. The actions were purposeful and direct and the results were explicitly attended to and evaluated. The design emphasis is on making the affordances of interaction unambiguous and available and ensuring that system feedback and state are clearly visible.

However, in many areas of human–computer interaction we have seen a growing number of systems and interaction paradigms where user attention and intention is lower. In CSCW the concept of awareness has been central for many years and similarly ambient interfaces emphasise low salience displays of background information. A number of terms have been used to refer to interfaces that include less explicit interactions: calm interfaces, tacit and implicit interaction. All emphasise output that is non-intrusive, and ecologically natural forms of input/control.

Whereas the traditional interface was based around controls and input devices, these low attention and natural input paradigms are more closely related to sensing technologies and contextual interpretations. Furthermore, human physiology may be sensed to influence interaction; for example the 2002 Commonwealth Games baton had an electronic 'flame' that flashed depending on the bearer's heart rate.

At the extreme end of the spectrum are *incidental interactions*, where an actor (user) performs an action for some purpose (say opening a door to enter a room), and the system senses this and *incidentally* uses it for some purpose of which the actor is unaware (perhaps adjusting the air conditioning), but which affects their future interactions with the environment or system.

In this section we'll look at some examples of incidental interaction and see how it fits within a spectrum of different levels of intention. We'll then see how this challenges major areas of traditional interaction design: the fundamental execution–evaluation cycle implicit in much of HCI, and the limits of our innate cognitive abilities. Finally we'll consider how to design and implement this sort of system, although these are still areas with no established best practice or standards.

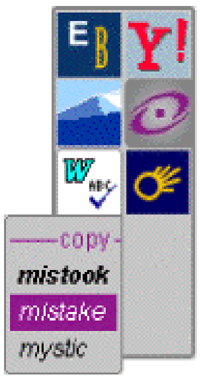
### 18.4.1 examples

Car courtesy lights operate differently depending on the model of car. They may turn on when the doors are unlocked or when the doors are opened. They may turn off after some fixed time, or when the doors are closed or the engine is started. Underlying this is some designer's model of the task of getting into a car – perhaps sorting out belongings in the car, looking at a map, etc., before setting off. The sensors are unreliable means of detecting the user's intentions, but the incidental interactions with the lights are designed to support the task. Note that the driver's *purpose* is to get into the car and *incidentally* the lights come on.

In the Pepys project at Xerox EuroPARC, all staff wore an 'active badge' that detected their location in the building using infra-red sensors [[WHFG92]]. At the end of each day, the Pepys system analysed the logs of people's location and used these to produce a personal diary for each person [[NEL91]]. Because Pepys knew about the layout of the offices, and who was where when, it was able to detect when two or more people were in the same location and create a diary entry for all of them, e.g. for Brian – "had meeting with Alison and Clarise ". Again, Alison and Clarise's *purpose* is to visit Brian's office and *incidentally* a diary entry is created for each of them.

The MediaCup [[GBK99,BGS01]] also facilitates incidental interaction. It is a base unit that can be attached to an ordinary coffee mug and detects movement (when drinking, walking around, etc.), pressure (for fullness), temperature (fresh coffee vs. old) and location. The information gathered by this gives some indication of the drinker's current activity and location, which can then be used for community awareness. Hans' *purpose* in filling the mug is to have a drink of coffee and *incidentally* his colleagues become aware he is taking a break.

Incidental interactions can also take place entirely within the electronic domain. In many electronic shopping sites the system keeps track of the items you have purchased or examined and then suggests additional products based on your inferred tastes. Your *purpose* is to buy product X and *incidentally* the system infers your tastes and suggests product Y.



One system some of the authors have worked on personally is onCue. This is an 'intelligent' toolbar that sits on the side of the user's screen. When the user cuts or copies any text onCue examines the clipboard. It analyses the content and, depending on the type of material, changes the tool items to reflect it. If the copied text is a post code, onCue suggests internet mapping services; if it looks like a name (initial capitals, etc.), onCue suggests internet directory services; if the text is a list or table of numbers, onCue suggests spreadsheet or graphing applications. As well as the triggering event being implicit, the 'suggestion' is deliberately low salience; the currently suggested services slowly fade in as small icons in the toolbar. Note again, the user's *purpose* is to copy the text somewhere else and *incidentally* potential useful services are offered. (See onCue case study at /e3/casestudy/onCue/)

### 18.4.2 the intentional spectrum

If we look back at these examples and think of related ones, we can see that they differ in just how 'incidental' the results are. If you get into the car and the courtesy light does not come on you may notice. Even though you didn't explicitly ask for the lights to turn on, you still *expect* them to do so.

In fact, there is a continuum of intentionality (figure 18.14). At one extreme are 'normal' intentional interactions such as pressing the computer key or pressing a light switch and expecting the light to come on. This would also include, for example, a gesture recognition system. Imagine a complex image recognition system that watches your hand movements so that you can point at a particular light bulb and say "light on" or "light off". Although this would be far from a traditional computer system, it is clearly intentional – the user wants the light on and deliberately does something that will have the desired effect.

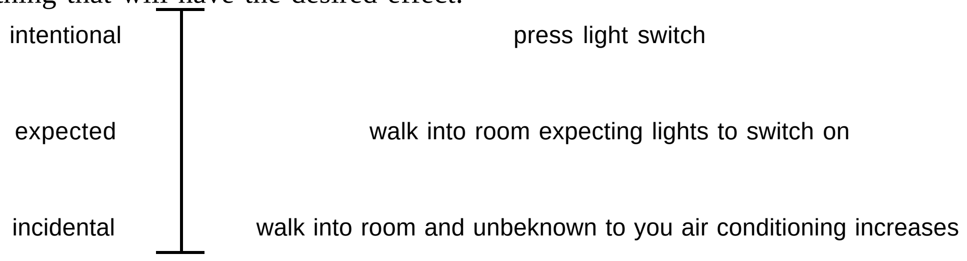

intentional
press light switch
expected
walk into room expecting lights to switch on
incidental
walk into room and unbeknown to you air conditioning increases

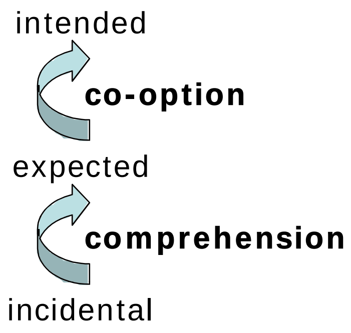
Figure 18.14 the continuum of intentionality

The automatic lights that are found in some public toilets fit somewhere in between. They are based on infra-red or ultra-sonic sensors that detect movement. If there is movement they come on; if not they turn off. These are more like the car courtesy light. When you walk in you expect the light to go on and would be unhappy if, as the door closed behind you, you found yourself in darkness.

The automatic central heating controller that detects who is in the room and adjusts the temperature accordingly is at the extreme end of incidental interaction. Unless the users consult the manuals in detail they may have no idea that this is happening behind the scenes. Certainly when they walk into the room it is no part of their model of what the act of entering means.

The automatic room lights are quite interesting as a small design change can turn them from an expected interaction to an incidental one. Imagine a house that has real light switches, but (for energy saving) always switches off the lights in rooms that no-one is in. When you approach the room the lights automatically turn on, so that you are never aware that they switch off. Fridge lights and doors that automatically unlock when an active badge is near are another example.

In addition, there may be changes between these caused by the user's understanding of the system. The continuum from intentional, through expected, to incidental interaction is largely about purpose – what the user thinks – not the actual system itself. Of course, certain types of system will suggest more or less strongly one mode of interaction, but there is some fluidity depending on the user's experience, awareness, etc. As users become more aware of the interactions happening around them they may move through the continuum towards more purposeful interaction.



- **comprehension**: incidental → expected – If users notice, consciously or unconsciously, that incidental interactions are taking place they may come to expect or rely upon them. For example, if you realise that the courtesy lights come on when you get into the car you may leave checking your route until you get in, knowing that the car will be lit then.
- **co-option**: expected → intended – When users know that something will happen when they perform an action, they may deliberately perform the action to cause the effect. For example, you may deliberately open and close the car door to trigger the courtesy light mechanism.

The opposite can happen as well. Imagine you use a gesture recognition system to open the door. Placing your palm open in front of you when you approach the door means "open the door now". After a while this action becomes proceduralised and you may no longer be conscious that you do it. For you it is as if the door always opens when you approach it. One day you approach the door, but you are carrying a box …

### *18.4.3 challenging our models*

As well as being an interesting interaction paradigm in its own right, incidental interaction really pushes our fundamental assumptions about interaction and our ways of modelling it. This will require a rethinking of HCI theory and practice more fundamental than that of the 1980s when GUI interfaces replaced character terminals.

#### interaction models

The explicit or implicit model behind nearly all interaction is some form of intentional cycle such as the Norman execution–evaluation loop [[N90]]. The user has some goal (intention), formulates some action that he/she believes will act towards that goal, performs the action and then re-formulates future actions based on the feedback.

In traditional cognitive modelling, this is seen as very plan-driven, with goal stacks, hierarchies, etc. In these accounts, the intentional cycle is seen as starting with the user, even to the point that the effects of feedback are often ignored. In more contextual accounts of interaction, such as situated action [[S87]] or distributed cognition [[H90]], the goals or intentions are more at the level of overall motivations or end-state aspirations. The focus tends to shift to a cycle of activity starting with the state of the world and recent system 'responses', with the user acting on the world in response to the current state. However, this is still clearly purposeful activity.

Incidental interaction and, to a lesser extent, expected interactions do not fit this picture. The user and system share the experience of the user's actions, but the purposeful activity of the user is distinct from the intended outcomes of the system.

This is not just a theoretical issue: it is an underlying assumption that cuts through nearly every usability guideline, principle and method. For example, the importance of rapid and visible feedback is based on the assumption that users need to understand fully the effect of their actions. In incidental interaction and low awareness applications the opposite is often true; feedback may be unobtrusive (and not explicitly noticed) or delayed (e.g. the heating level slowly changing). Even expected interactions are more likely to be noticed when they don't happen than when they do.

## cognition

Natural *inanimate* physical things have a set of properties intrinsic to their physicality:

- directness of effect – You push something a little and it moves a little, you push hard and it moves a lot.
- locality of effect – Effects are here and now. If you pushed a rock and then two seconds later it moved you would be disturbed.
- visibility of state – Solid objects have a small number of relevant parameters that define their dynamic state: location, orientation. We have some difficulty with invisible properties such as velocity and even more when physical things do have hidden state, for example the 'joke' balls that have a ball bearing inside and so do not move in a straight line. Of course this example is not natural but constructed.

We have evolved as creatures to cope with physical things and other creatures, not technological devices. Although we have higher level reasoning that enables us to cope – the same reasoning that enables us to create technology – this is only significant when we 'think about' things; our more innate cognitive abilities are shaped by the natural.

Computer systems (and other complex technology such as electrical and pneumatic systems) break these intrinsic properties of physical objects. Computation creates complexity of effect, networks introduce non-locality in space, memory non-locality in time, and a computer has a vast number of invisible variables in its hidden internal state.

We cope (just) with this, because either we rationalise and use higher-level thinking to make sense and to make models of these complex non-physical interactions, or we treat the computer as animate. In addition, one of the reasons for the development of the GUI interface style is that it makes the electronic world more like real (inanimate) things.

In incidental, expected and low awareness interactions the design is such that the user is not paying attention to, or is unaware of, the system's activities. That is, we are not able to bring our higher cognitive abilities to bear and are dependent on our more innate intelligences, which are of course ill prepared for unphysicality. To make matters worse, the system activities are often triggered by physical actions and movements of the user and are manifest in the physical world. In a computer system we are able to re-frame ourselves to expect odd or apparently magical actions to occur. In the real world these are deeply disturbing.

### *18.4.4 designing for incidental interaction*

Traditional task analysis is also highly purposeful, although debatably less wedded to this than cognitive models. Certainly to cope with more contextual interactions task analysis methods need to evolve to include or link to representations that are more about the physical world and the rich ecology of lived work or domestic life. In the previous section we examined some of the potential issues and extensions that may be necessary for this.

However, incidental interaction poses a more fundamental question – what task do we need to model? In incidental interaction we have two 'tasks' that are occurring.

(i) the user's primary task – their purposeful activity.

(ii) the task that the incidental interaction is attempting to support or achieve

Often, as in the case of the courtesy light, the two are the same task, but it is used in different ways. The user's purposeful activity is assumed to occur, to a large extent, independently of the system's actions. We need to model it in order to computationally interpret the user's actions as activity. In contrast, we need to model (ii) in order to understand how to facilitate or progress it.

In addition, low intention and sensor-based systems often include uncertain inferences. In traditional interfaces there is intended to be no ambiguity – the user presses the 'x' key and an 'x' appears in the document. Of course the user may have mistyped, or may not realise the system is in a mode where 'x' means 'exit', but these are 'errors' or misdesigns; if all is going well there is no ambiguity. In contrast, a sensor-based system may 'think' that you are resting because you are not moving about much and turn down the music volume, but you may simply be sitting still listening to the strident sounds of Beethoven's Fifth. Happily the things controlled or intended to be controlled by these interactions are often less critical. You might like the heating a little warmer or colder, but it is not absolutely essential whether the system gets it right.

At the time of writing there are no developed methods for dealing with these low intention interactions, although there are some proposals [[DGN02, SA03]]. However, we can begin to see how more traditional methods may change to accommodate these new interaction styles.

Clearly any design for low attention must identify two things:

- **input** – what is going to be sensed (e.g. body heat, pressure pad)
- **output** – what is going to be controlled (e.g. light, heating)

Once we know these we can look at scenarios or task models and label these to see what we would like to happen at each step. In some cases there will be a definite requirement (e.g., the car courtesy lights must not be on when the car is moving), in others there may be simply a desire (e.g., it would be good for the light to be on to put the key in the ignition). Of course, the steps in the task or scenario may involve user intentions or other aspects of context not immediately available to the computer system – that is why we need to be able to infer this context. To do this we can look at the available sensors and see how certain we can be of the current context based on their data. This can then be used both to verify whether we can be certain of the context at points at which there is a definite requirement (and add explicit controls if not), and also to be able to control the output at other times so that the users usually (but not always) get what they want.

**DESIGN FOCUS**
**Designing a car courtesy light**

The first step in designing a sensor based system is to workl out what you would like to control. In this case the interior light. Next we look through some scenarios and label steps in the scenario. At each stage we not whether the ligjts should be on (more +s) or off (more –s). Here is one such scenario:

| | | | |
|---|---|---|---|
| 1. | deactivate alarm | 0 | |
| 2. | walk up to car | 💣 | safe, advertises presence |
| 3. | key in door | – | |
| 4. | open door and remove key | + | |
| 5. | get in | + + | |
| 6. | close door | 0 | |
| 7. | adjust seat | + | |
| 8. | find atlas | + + | |
| 9. | look up route | + + + | |
| 10. | find right key | + | |
| 11. | key in ignition | – | |
| 12. | start car | 0 | |
| 13. | seat belt light flashes | 0 | |
| 14. | fasten seat belt | + | |
| 15. | drive off | – – – – | illegal ! |

Step 2 is interesting, if you ask different peoiple you get different responses. Some like to see the lights go on when the alarm is disarmed. However, others fear it advertises their presence and leaves them vulnerable to attack.

Note too that we have assumed the alarm remote control does not actually unlock the car. This was partly so that we could have step 3 where the lighted interior makes it slightly more difficult to put the key in the car door.

At this stage we can either work a full task analysis and mark this up similarly so that for each task and subtask has a desired lighting attached. Alternatively we could move forward to more detailed design.

See web site for full case study: `/e3/casestudy/car-lights/`

### *18.4.5 implementing sensor-based systems*

In incidental interactions it is very likely that sensors are not used solely for their original purpose. This suggests the need for quite open architectures. For example, onCue uses a very open blackboard-style architecture for exchanging information between self-discovering and self-configuring components [[DBW00]]. Unfortunately, at the level of individual applications it is far harder to get contextual information without writing special code for each potential application. This is one of the reasons for using copy/cut to the clipboard as the main trigger – it is one of the few cross-application standards.

Furthermore, many of the contextual interactions envisaged in this area occur in domestic or other private environments. If we are not careful architectural openness could violate privacy – imagine if the can of beans (with intelligent food label) you just bought communicated back to the manufacturer the contents of your food cupboard.

Highly contextual interactions must also take on board the fact that many of the most important phenomena are not events (things that happen at specific moments), but status (things that always have some measurable value). Status–event analysis highlights common phenomena that can be used to understand such systems, but this also impacts on the underlying system architectures [[D91,DA96]].

Although there are many research and commercial systems being produced using sensors, there are no clear 'standard' architectures like the Seeheim model or MVC developed for traditional interfaces. However, there are some features that are likely to be present in many systems (see figure 18.15).

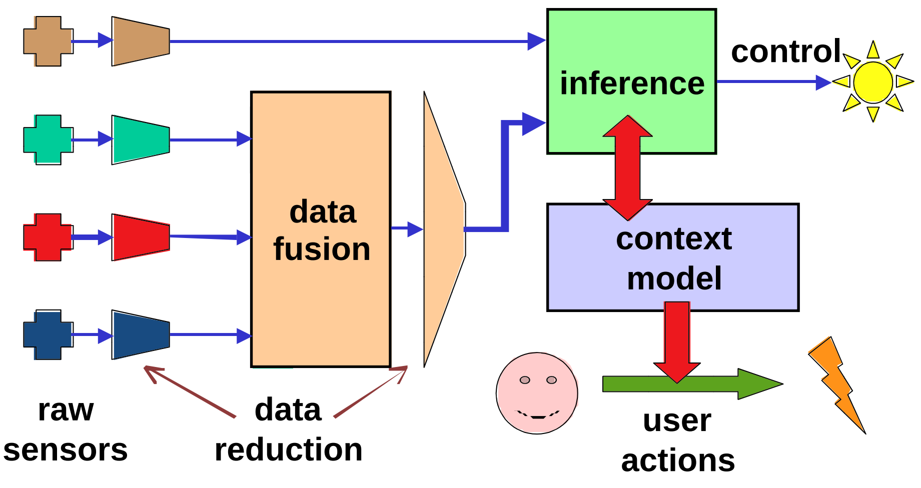


Figure 18.15 Potential architecture of sensor-based system

Some sensor-based systems may employ quite simple sensors, for example the door open/closed sensor for car courtesy lights. However, where the raw sensors are capturing richer data it may well be that there is too much data to process fully. In these cases the sensors may have to somehow filter or pre-process their outputs before passing on their data. For example, the MediaCup senses the temperature of the cup and pressure on the bottom of the cup, and has ball bearing sensors to give approximate tip in two x–y directions; all of which could be sensed many times per second and at high resolution. However, only a small bit-mask with indicators such as hot/cold, moving/still is sent via the infra-red link to the network.

Often the results from several sensors may need to be processed together to give a usable output. For example, several heat sensors may be averaged. This is a form of *data fusion* – bringing together multiple data sources to build a more accurate picture. This data fusion stage may also reduce the information; for example, the single average from 10 individual temperature readings.

These processed sensor readings are then used to drive some form of inference. This may be a few hand-coded rules: "when Alison's MediaCup is moving she is in the office"; more sophisticated rule-based systems; or some form of neural network. This inference will typically interact with some form of model of the users' context built up over time. For example, if in the past the ultrasound sensors have detected movement in a room and the pressure sensors under the doormat have not been triggered, then the system 'knows' there is likely to be someone there, even though the ultrasound sensors currently detect no movement.

Finally this contextual information has to be used. It may be used directly to drive some controlled output, for example, the lights in the car or room heater. Alternatively it may be used to modify the effects of users' actions based on the inferred context. For example, depending on who is believed to be in the house and the time of day, the TV may default to different channels when it is turned on.

## 18.5 Summary

Real interaction is not simply pressing a button and seeing something happen on screen. In this chapter we have looked at some of the ways to model and understand rich aspects of interaction.

We looked at status–event analysis, an ‘engineering’-level technique that encompasses formal methods, semi-formal analysis and naïve psychology, allowing us to consider issues that bridge system and user behaviour. Whereas most formal notations focus on the state changes occurring at particular moments (events), status–event analysis puts equal weight on status phenomena, such as computer screens, which always have a value. Important properties of status–event descriptions include the difference between actual and perceived events, polling to discover status change and the granularity of time. Timeline diagrams showing events and status in human–computer interaction allowed us to examine the delays in notification of email arrival and errors using on-screen buttons.

In section 18.3 we looked at the way existing task models could be extended to encompass rich contexts including other people and physical artefacts. Several existing techniques including CTT and GTA already include ways to allocate subtasks for different human and machine roles. In contrast, few methods deal well with the information required at each stage although this is not difficult to add. In order to keep tasks operating in the right order at the right time we saw that physical artefacts as well as other external events act as triggers that make things happen when they happen, and placeholders to implicitly record where people are in a process.

Finally we looked at the way that some recent ubiquitous computing applications have radically new modes of interaction. We considered a continuum based on the level of intentionality. At one extreme were traditional intentional acts. At the other extreme was incidental interaction where the user acts for one purpose and incidentally the system interprets the action to aid or help the user. In between is behaviour that the user does not explicitly request, but which is expected to occur. These low intention interactions cannot easily be understood within standard models of interaction. As yet there are no established design techniques or implementation architectures, but we saw that there are promising early methods.

### Exercises

18.1 Can you suggest any improvements to the screen button feedback problem discussed in Section 18.2 that would distinguish at the interface between the two cases of hitting or missing the button? Is there any guarantee with your solution that the user will notice the distinction?

18.2 Brian wants to make a dinner date with Alison. He knows she will not be able to read email, as she is away for a few days, and he doesn't have her hotel number. He types and prints a letter, which he puts in her pigeonhole. Alison's secretary always checks the pigeonhole several times a day, and when she finds the letter she reads it and rings Alison and tells her.

Analyse this story using a status–event description.

18.3 Look again at the tea making task analysis in Chapter 15 (figure 15.3). Go through this and look for triggers and placeholders. You will need to make assumptions (e.g. is the kettle the kind that whistles when it boils?) so document these.

18.4 Rank the following in terms of levels of intention or consciousness

automatic doors into hotel, automatic water taps in wash basin, reversing lights in a car, ultrasonic burglar alarm, auto-numbering lists in a word processor, web page counter, font menu in word processor that shows recent fonts at the top of the list

If you have a group, you could each rank them separately and then discuss your answers.

Why are some more consciously considered than others?

Think of more examples.

## Recommended reading

- Dourish, P. (2001). Where the Action Is: The Foundations of Embodied Interaction. MIT Press.
  Philosophical and practical discussion of what Dourish calls 'embodied interaction' that is interaction taking place in a real physical and social context.
- F. Paternó. Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, Heidelberg . 2000.
  A full design methodology and approach focused on the use of ConcurTaskTrees. Note that the collaborative aspects of CTT were developed after this book.
- A. Dix, D. Ramduny-Ellis, J. Wilkinson. Trigger Analysis – understanding broken tasks. In The Handbook of Task Analysis for Human-Computer Interaction. D. Diaper & N. Stanton (eds.). Lawrence Erlbaum Associates, 2003 (in press)
- Detailed treatment of triggers and placeholders discussed in section 18.3.
- B. Nardi and V. O'Day Information Ecologies – Using Technology with Heart. MIT Press, Cambridge, USA, 1999
  A beautiful book probing the rich interactions between people and technology. It seeks to find a middle way between those who abhor technology and those who adore it. The majority of the material in the book is structured around real case studies of systems in different settings from schools to hospitals, libraries to offices. It is full of vignettes like the researcher who failing to make conversation with a group of school children sits down at a terminal near them and instantly becomes accepted within their on-line community.

•

## New References for this chapter

[[BGS01]] Beigl, M., Gellersen, H.-W. and Schmidt, A. MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects. Computer Networks, Vol. 35, No. 4, Special Issue on Pervasive Computing, Elsevier, March 2001, pp. 401-409.

[[D91]] A. J. Dix (1991). Status and events: static and dynamic properties of interactive systems. Proceedings of the Eurographics Seminar: Formal Methods in Computer Graphics, Ed. D. A. Duce. Marina di Carrara, Italy.

[[DA96]] A. Dix and G. Abowd (1996). Modelling status and event behaviour of interactive systems. Software Engineering Journal, 11(6) pp. 334-346.

[[DBW00]] A. Dix, R. Beale and A. Wood (2000). Architectures to make Simple Visualisations using Simple Systems. Proceedings of Advanced Visual Interfaces - AVI2000, ACM Press, pp. 51-60.

[[D02]] A. Dix (2002). Managing the Ecology of Interaction. Proceedings of Tamodia 2002 - First International Workshop on Task Models and User Interface Design, Bucharest, Romania, 18-19 July 2002

[[GBK99]] H-W. Gellersen, M. Beigl, H. Krull (1999). The MediaCup: Awareness Technology embedded in an Everyday Object. Int. Sym. Handheld and Ubiquitous Computing (HUC99) Karlsruhe, Germany, 1999

[[H90]] Hutchins, E. (1990), The Technology of team navigation. In *Intellectual teamwork: social and technical bases of collaborative work.* Gallagher, J., Kraut, R. and Egido, C., (eds.), Lawrence Erlbaum.

[[NEL91]] Newman, W., Eldridge, M. & Lamming, M. (1991). Pepys: Generating Autobiographies by Automatic Tracking. In Proceedings of the second European conference on computer supported cooperative work – ECSCW '91, 25-27 September 1991, Kluwer.Academic Publishers, Amsterdam. pp 175-188.

[[N90]] D. Norman (1990). The Design of Everyday Things. Doubleday, New York.

[[S87]] Suchman, L. (1987). Plans and Situated Actions: The problem of human–machine communication. Cambridge University Press,.

[[WHFG92]] Roy Want, Andy Hopper, Veronica Falcao, Jonathon Gibbons. The Active Badge Location System. ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992, pp 91-102.

[[VW00]] Van der Veer, G.C. & Van Welie, M. (2000). Task Based GroupWare Design: Putting theory into practice. In *Proceedings of DIS 2000*, New York, United States.

[[BM95]] Barnard, P. and J. May (1995) Interactions with Advanced Graphical Interfaces and the Deployment of Latent Human Knowledge. In *Proc. of Design, Specification and Verification of Interactive Systems, DSVIS'94* . F. Paternó (ed). Springer.

[[BBS95]] Bowers, J., G. Button and W. Sharrock (1995). Workflow from within and without: Technology and cooperative work on the print industry shop floor. In *Proc. of ECSCW'95*, Kluwer.. pp 51–66.

[[DHW00]] Dearden, A., M. Harrison and P. Wright (2000). Allocation of function: scenarios, context and the economics of effort. *Int.l J. of Human-Computer Studies*, **52**(2):289-318.

[[D89]] Diaper, D. (1989). Task Analysis for Knowledge Descriptions (TAKD); the method and an example. In *Task Analysis for Human-Computer Interaction*. D. Diaper (ed.), chapter 4, pp. 108-159. Ellis Horwood.

[[DRW03]] A. Dix, D. Ramduny-Ellis, J. Wilkinson (2003). Trigger Analysis - understanding broken tasks. In The Handbook of Task Analysis for Human-Computer Interaction. D. Diaper & N. Stanton (eds.). Lawrence Erlbaum Associates, 2003 (in press)

[[D02]] Dix, A. (2002). *Incidental Interaction*. available online: http://www.hcibook.com/alan/topics/incidental/

[[D01]] Dourish, P. (2001). *Where the Action Is: The Foundations of Embodied Interaction*. MIT Press.

[[HP90]] Howes, A., and S. Payne (1990). Display-based competence: towards user models for menu-driven interfaces. *Int. J. of Man-Machine Studies*, **33**:637-655.

[[HORSR95]] Hughes, J., J. O'Brien, M. Rouncefield, I. Sommerville and T. Rodden (1995). Presenting ethnography in the requirements process. In *Proc. IEEE Conf. on Requirements Engineering, RE'95*. IEEE Press, pp. 27–34.

[[H90]] Hutchins, E. (1990), The Technology of team navigation. In *Intellectual teamwork: social and technical bases of collaborative work.* Gallagher, J., Kraut, R. and Egido, C., (eds.), Lawrence Erlbaum.

[[KP85]] Kieras, D. and P. Polson (1985). An approach to the formal analysis of user complexity. *Int.l J. of Man-Machine Studies*, **22**:365-394

[[LPV01]] Limbourg, Q., C. Pribeanu and J. Vanderdonckt (2001). Towards Uniformed Task Models in a Model-Based Approach. *In Proc. DSV-IS 2001*. Springer pp.164-182

[[NEL91]] Newman, W., M. Eldridge and M. Lamming (1991). Pepys: Generating Autobiographies by Automatic Tracking. In *Proc. of ECSCW '91*. Kluwer... pp 175-188.

[[P99]] Paternó, F. (1999). *Model-based design and evaluation of interactive applications.* Springer.

[[S87]] Suchman, L. (1987). *Plans and Situated Actions: The problem of human–machine communication*. Cambridge University Press,.

[[W89]] Walsh, P. (1989). Analysis for task object modelling (ATOM): towards a method of integrating task analysis with Jackson system development for user interface software design. In *Task Analysis for Human-Computer Interaction*. D. Diaper (ed.), pp. 186-209. Ellis Horwood.

[[WF86]] Winograd, T. and F. Flores (1986). *Understanding computers and cognition : a new foundation for design.* Addison-Wesley.

[[SA03]] Sheridan, J.G., Allanson, J. PICK - A Scenario-based approach to Sensor Selection for Interactive Applications. Proceedings of the 10th International Conference on Human-Computer Interaction (June 22-27 2003) Crete Greece.

[[DGN02]] E. Dubois, P. Gray, and L. Nigay (2002). ASUR++: A Design Notation for Mobile Mixed Systems. Mobile Human-Computer Interaction, Proceedings of the 4th International Symposium, Mobile HCI 2002. F. Paternò (Ed.). Springer-Verlag LNCS 2411, pp. 123–139.