Finding Out
event discovery using status–event analysis

Alan Dix

School of Computing, Staffordshire University, UK

http://www.soc.staffs.ac.uk/~cmtajd/topics/status/

This paper uses status–event analysis to systematically decompose the process by which the knowledge and effects of events are propagated through a system. The analysis focuses on events – things that happen; status – things that are always there; and agents – affected by events and interacting with each other and status phenomena. Single interactions between these are characterised by the source of event knowledge, the initiator of interaction and the trigger for interaction. This generates a structured set of alternatives by which events are transmitted through chains of interaction. The application of this analysis to notification server design is briefly described.

keywords: status–event analysis, delays, causality, CSCW, notification

1. Introduction

Status–event analysis is the name of a collection of methods developed and used over the last 10 years to understand and analyse various issues in both single-user interfaces and CSCW. These methods are unusual in that they treat events (things that happen) and status (things that are) on an equal footing. The power of this descriptive framework has been that it is possible to describe human, compute and physical phenomena in the same framework and furthermore see common issues and phenomena arising.

Status–event analysis (S–Ea) has proved useful in a number of areas from the design of fine-grained interaction in auditory interfaces [5-7, 15] to the construction of software architectures for application integration [43, 44]. In particular, it has recently been used in order to elucidate the design space for notification servers within CSCW systems [39] and is currently being used as one of the major analysis techniques in a project investigating the construction of mobile, multi-user, multimedia applications[1] where issues of context-awareness demand the explicit representation of status phenomena [41].

Several uses of S–Ea have focused on time delays and in particular the importance of mediation, both where status phenomena may be used as a means of communicating events (for example highlighting an icon when email arrives [18, 19]) and also where events are used to preserve a desired relationship between status phenomena (as in constraint maintenance systems [28, 33]). In previous work a rough typology of status–status mediation (e.g. how when an icon is dragged

---

1 "Interfaces and infrastructure for mobile multimedia applications", funded by the EPSRC MNA programme . http://www.hiraeth.com/alan/projects/I2M2A98

following the mouse position) was used to expose typical timing problems [13], but in early work this consisted of a rather unstructured list of options and it was unclear whether the list was complete. Although the focus of this previous work was on status–status mappings, the key issue was event discovery – how an agent can find out when an event has occurred.

In this paper the space of mediation options is analysed in a structured fashion giving rise to a multi-dimensional taxonomy of mechanisms in which events can be discovered by agents within a status–event framework. The results of this work has already been invaluable in the analysis of notification mechanisms, but this application is only described briefly as it is dealt with in detail elsewhere [39].

The paper will begin with a review of status–event analysis leading into the structured analysis which focuses on three main aspects of event discovery: source of event information, the initiative in finding out and the triggering events which prompt discovery. By mapping out chains of simple interactions we see behaviours of event discovery such as demand and data driven discovery. We will then look at the way this work has helped in the analysis of notification server design. The paper concludes with an examination of related work within user-interface implementation and specification literature, and more widely.

2. What is status–event analysis?

Events happen at a specific moment whereas status refers to phenomena which may be measured or sampled at any time such as the current air temperature, the appearance of a screen or the position of a mouse. Many formal and informal analysis methods attempt to describe all phenomena in terms of one or the other. In particular, the discrete nature of computer systems has tended to mean that all phenomena are cast as events. Elsewhere the danger of this approach has been argued as it means that specifications are both unnatural (they don't easily describe the required phenomena) and over-specific (they don't accurately describe the phenomena). Status-Event analysis consists of methods which take this distinction seriously.

One way to see this distinction is to consider traces of activity for status compared with those for events. For an event phenomena, the trace is a sequence of times the event happens together with any value associated with the instance of the event:

$$\text{etrace: Event} \;\nrightarrow\; \textbf{seq}\;(\,T \times \text{Eval}\,)$$

$$\text{where } \forall\, i, j \in \textbf{dom}\,(\text{etrace}(e)) \;\bullet$$

$$(\; i < j \;\wedge\; \text{etrace}(e)_i = \langle t_i, v_i \rangle \wedge\; \text{etrace}(e)_j = \langle t_j, v_j \rangle \;) \;\Rightarrow\; t_i < t_j$$

In contrast a status phenomena always has a value and so a status trace gives a value for each moment in time:

$$\text{strace: Status} \times T \;\longrightarrow\; \text{Sval}$$

Status–event analysis was first used to model mouse dragging in single user interfaces and some aspects of shared interfaces [18], but its wider application soon became evident. One early application was the analysis of delays in email delivery using timeline techniques [16, 19, 20]. Figure 1 shows the individual timelines for

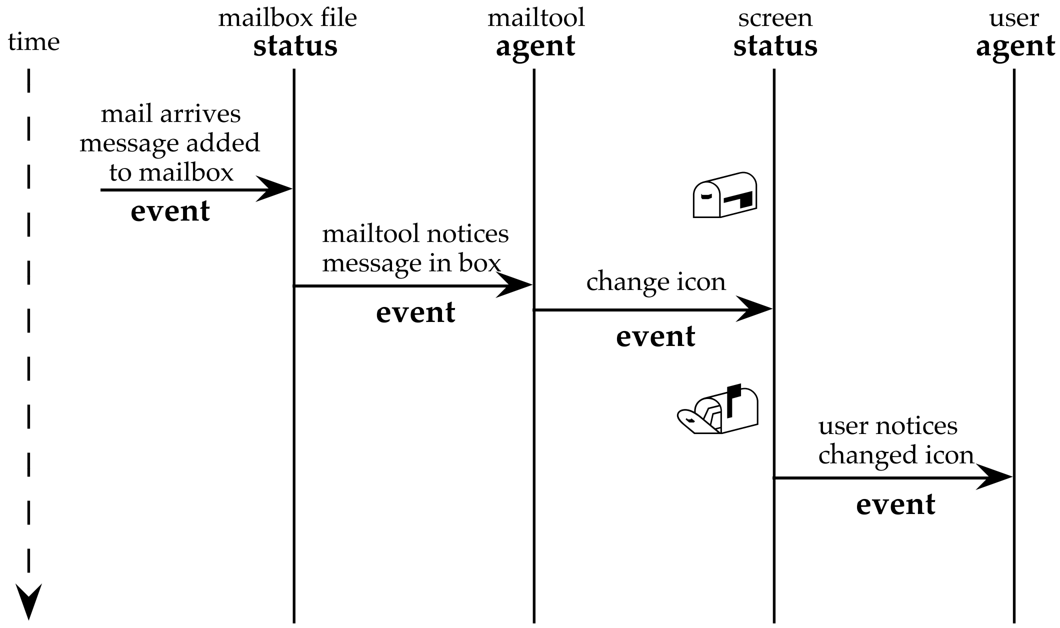

Figure 1. Status–event timelines for email delivery

several agents and status phenomena within the system. This example highlights several critical features that were subsequently found in many systems.

- *mediation* – The file system (a status) acts as a mediator between the agent that actually receives email (e.g. sendmail) and the user's email agent – a computer–computer interaction. Similarly the email agent highlights an icon (status) and uses this to signal the arrival of email to the user – a computer–human interaction. In each case one agent informs another of an event by changing a mediating status.
- *status–change events* – Although the original event is that email arrives, the event that is significant for the mailtool and user is the relevant status-change event – the change in the file systems, or the change in the icon.
- *polling* – In each case the relevant agent finds out that the status has changed by intermittent checking of the mediating status; that is polling.
- *actual and perceived events* – There is an actual change in status when the new message is added to the mailbox, but the mailtool does not perceive this event until it next polls the file system. Similarly, there is an actual change in the screen status when the icon is changed, but the user may not notice this straight away.

Notice how similar techniques and issues arise during human–computer and computer–computer interaction.

S–Ea has since been used by the author and others in a range of situations using both formal and informal analysis techniques.

As a semi-formal analysis tool S–Ea has been used to aid the design of auditory interfaces. When considering the addition of sound to on-screen buttons, S–Ea both guided the appropriate choice of auditory feedback and also helped design experiments which induced normally infrequent expert slips within a short experimental setting [6, 15]. Because of the importance of mode (a special kind of status) in user interfaces, Brewster considered this alongside events and status and applied the resulting event–status–mode (ESM) technique to various auditory interfaces including the auditory–enhanced scrollbar [5, 7].

Again as a semi-formal framework, S–Ea has been one of the theoretical bases of an analysis of long-term interaction – processes which take place over hours, days or months. In particular, it helped focus on the role of environmental cues (pieces of paper on desks, Post-It Notes) as triggers for action [17, 21].

S–Ea has also been used by Wood as the foundation for Cameo an agent-based software architecture implemented in Java, which has been used to construct CyberDesk an interface for context-aware application integration [43, 44]. The actual programmes written under Cameo operate in an event-based fashion, but Cameo embodies the spirit of S–Ea by giving a structured way of mapping status into events. Cameo agents can advertise 'nodes' and allow other agents to 'manipulate', 'observe' or 'examine' the node (or any combination). Manipulation allows the value can be altered externally, observation allows the setting of callbacks (handled by Cameo) and examination allows polling behaviour.

The author's own work with Abowd has looked more closely at notations and formal implications of S–Ea including the specification and analysis of shared scrollbars in collaborative systems [1, 14]. Most important for this paper was the analysis of delays due to status–status mediation [13]. We will review the situation at that point and see how a more detailed analysis is needed.

Consider, a simple functional relationship between two status values 'x' and 'y':

$$y = f(x)$$

First note that in a digital system it is impossible to permanently maintain such a status–status mapping unless either they are linked physically, or the 'y' is merely a virtual value calculated from 'x' when needed. With these exceptions changes to y will always lag a little behind those to x as the system attempts to 'catch up'.

If this is the case, how does 'y' actually get changed? In [13]. we listed four alternatives:

(i) the agent which changed ‘x’ also changes 'y'.

(ii) an agent responsible for ‘x’ (active value) changes 'y'.

(iii) an agent responsible for ‘y’ polls the value of 'x'.

(iv) an agent responsible for the mapping (e.g. constraint system) polls 'x' and the updates 'y'.

Although this seems like a reasonably complete list there are clearly extra cases involving other intermediaries. Furthermore, how do we know if we added cases whether those made it complete or whether more cases again need to be considered?

The next section answers this by considering the process of event discovery systematically.

3. The conceptual world of status–event analysis

In this section we will first examine in more detail the things that make up the world when viewed through the S–Ea framework.

3.1 Fundamental entities

S–Ea obviously emphasises status and events – these are the observable phenomena around us. However, in the analysis so far agents have also been mentioned several times. Agents, whether computational or human, are the subject of or may generate events and may observe and change status phenomena.

Clearly, some status phenomena are external to any identifiable agent, for example, the pattern of light shining through the clouds. Other status phenomena are closely connected, being the external expression of some part of an agent's state, for example, the expression on someone's face.

Status and agents are similar in they both persist through time and furthermore agents will often posses an internal state which is itself a form of status. Because of this, the formal specification notations used previously for S–Ea have regarded status as an attribute of agents [1, 14]. The representation of status in Cameo [43] and in the author's own early work on S–Ea-inspired software architectures [12] have both had a similar flavour – embodying status within agents which manage the status and inform interested third party agents of status change.

In an implementation setting this identification makes sense, but for this analysis it is better to keep the two separate and so status and agents are regarded as distinct. Thus we will be looking at a three way relationship between status, events and agents.

3.2 Events

Events in a S–Ea world can either arise internally from agents (A→) or status (S→), or externally (X→). In addition, in both the electronic or physical world, messages or signals are sent which take some time to arrive due to the properties of the channel or medium. The receipt of such a message is clearly caused by the sending of the message, but is a distinct event (M→).

External events (X→) are least interesting, they simply happen as input. Similarly message receipt (M→) is only of one kind. Because we are considering agents and status separately, the status phenomena have no independent initiative and so the only kind of event from these are status-change events (S→).

This leaves us with the case of events emanating from agents (A→) – that is, when they **do** something. An agent may respond due to some previous **trigger** event. This may be effectively instantaneous, or more normally after a delay due to computation, thinking or a timed interval. Agents also have initiative and hence may generate events without explicit external triggers. These may either be apparently causeless sporadic events (normally human agents) or may be due to periodic or timed behaviour (normally computational agents).

In summary, we have the following kinds of event:

- external (X→)
- status-change (S→)

- message receipt (M→)
- dependent (→A→)
  - instantaneous, after computation/thinking or after timed delay
- independent (A→)
  - sporadic, periodic or at set time

4. Simple Interactions

An event E has happened somewhere. How does an agent A get to know that it has happened? In other words, how does the actual event E become a perceived event for A? A similar question arises for a status S. How does the event E get to affect the value of S?

In practice there may be a whole chain of events and interactions between E and A or S. In section 5 we will look at these chains,. However, in this section, we will begin by looking at one step in that chain , a simple single-stage interaction.

Imagine an entity (agent or status) has already been influenced by the event E. We want to see how another entity can interact with it to in turn be influenced. We will write the participating events, status or agents left-to-right denoting the direction of causality between them as the event E is propagated.

4.1 External events

The simplest case is when the event is an external an agent or status is directly affected by an external event. We will write these:

- X→A external event influences an agent
- X→S external event modifies status

4.2 Source and target

Looking at single interactions between agents and status we get four main possibilities depending on whether the source and target of the interaction are agents or a status:

- A→A communication between agents
- A→S agent modifies status
- S→A agent notices status change
- S→S physical link between status

4.3 Delays and synchronicity

Usually the interactions leading to a status change (A→S and S→S) are synchronous – when the agent changes the status value, it instantly changes, when two status are physically linked, they move together. However, if either of these are introduced in a high-level specification they may latter be implemented by several stages hence delays may occur.

In the cases where agents are affected by events there are frequently delays. In the case of communications between agents some sort of message will be sent which may both take time and be unreliable (A→M→A). We will not represent these message explicitly further in this paper, but simply note that any A→A interaction may be subject to such delays. In the case of status change, there may also be a gap between the actual change and the perceived event for the agent.

Note that where S→S links are really physical processes there may not be a finite number of distinct change events, but instead a continuous linkage between the two, possibly involving complex dynamic behaviour.

4.4 Initiative

When the target of an interaction is a status there is no question of where the initiative lies. However, when the target is an agent we need to know whether the source initiated the interaction or whether the target agent chose to do so.

Consider first A→A interactions. In the next room my daughter is watching the England–Argentina World-Cup Match. A goal is scored. Two thing may happen. My daughter might come through and say "England has scored". Alternatively, at a suitable break in the typing I might go through and say "has there been a goal?". In both cases the knowledge of the goal scoring event has gone from my daughter to me, but in the latter case the initiative came from me, the target. We will distinguish the two case by reversing the arrow, so that causality always flows from left to right, but the arrow direction denotes initiative:

- A→A source agent **tells** target agent
- A←A target agent **asks** source agent

With S→A interactions, one could argue that the initiative must always come from the agent who must watch the status:

However, in physical systems status phenomena can force themselves on our attention. Think of the jungle film where the heroine wakes up tense because the sounds of the jungle have ceased. In computational system s we get a similar effect when there is some sort of 'gatekeeper', a primitive agent which can inform other agents of status changes (but have no independent initiative). Examples of this include active values [10] and some databases with triggers.

- S→A **gatekeeper** of status tells target agent
- S←A target agent **watches** source status

4.5 Trigger

Finally, we need to know what event triggered the interaction.

In the case when the status is the source *and* initiative (S→S and S→A) then this can only be the event which modified the status. However, the cases where source or target agent has the initiative (A→S, A→A, A←A and S←A) are more complex. Recall there were two main types of agent initiated events: dependent (triggered by previous event to agent) or independent (sporadic, periodic or at set time).

It is often, the case that source initiated interactions will be dependent, triggered by the event E. This is a form of data driven computation. In contrast, target initiated interactions often involve sporadic or periodic polling. However, all four cases can arise. Some mail systems gather several mail messages together in a spool file and then periodically send all the accumulated mail, a form of periodic update. This is a periodic source initiated interaction. Also, the target agent may poll due to some other event. In particular, this event may be the request for information from a further agent – demand driven computation.

	dependent (previous event)	**independent** (sporadic, periodic etc.)
source	data driven	periodic update
target	demand driven	polling

5. Causality chains

As already noted, we can see that there may be a chain of interactions that lead to an agent or status being influenced by an event E. We can use the notation from the previous section for these. Consider the chain:

$$X \rightarrow A_1 \rightarrow S_1 \rightarrow S_2 \leftarrow A_2$$

An external event X initiates this. It affects the agent A_1 which reacts by modifying the status S_1. Some physical process (or constraint management) links S_1 and S_2, so S_2 is changed as well. Then due to periodic or sporadic polling, agent A_2 looks at S_2 and so becomes aware that X has happened.

At each link in this chain we can examine the triggering events, consider delays etc., following the analysis of the previous section. In particular, this can give us the sort of delay analysis used in the status–event timeline diagrams.

Looking at such chains we can see structures such as demand-driven discovery:

$$A \leftarrow A \leftarrow A \leftarrow A$$

or status mediation between communicating agents:

$$A \rightarrow S \rightarrow A$$

So, where does the chain start?

If the event of interest is E, then this may be the start point as in the case of the external event above. However, the chain may also start at some common cause. For example, suppose an agent A_X is interested in status change for S_Y It may find out by observing for direct effects of S_Y:

$$S_Y \rightarrow A \rightarrow A \rightarrow S \rightarrow S \leftarrow A_X$$

However, there may instead be a second status S_Z which changed and separate chains of events gave rise to the status change of S_Y and the event perceived by A_X:

$$S_Z \begin{array}{l} \nearrow A \rightarrow A \rightarrow S \rightarrow S \leftarrow A_X \\ \searrow A \rightarrow A \rightarrow S_Y \end{array}$$

Note that this is predictive discovery. If the propagation down the lower chain is slower than the upper, or unreliable then A_X may think that the status changed has occurred before it actually happens, or even when it never happens at all!

Although this seems a rather extreme case it is quite common when the lower chain is trivial. For example, consider the chains:

$$A_Z \begin{array}{l} \nearrow A \rightarrow A \rightarrow S \rightarrow S \leftarrow A_X \\ \searrow S_Y \end{array}$$

In this case, the agent AZ, which caused the change to SY also sets in motion a chain of events which results in AX perceiving the status change. This is common in computational systems. For example, a program may change an internal data structure and then update the screen. When you see the screen change you will regard it as an indication of the changed data:

$$A_{program} \begin{array}{l} \nearrow S_{screen} \rightarrow A_{user} \\ \searrow S_{data} \end{array}$$

6. Application

This analysis of cases is quite tortuous at times. However, it pays off by giving a much greater sense of certainty in the completeness of the resulting analysis. This has already proved powerful in the analysis of the design space for notification servers [39].

We consider several client applications (agents) running on each participant's machine. These clients update shared data (perhaps in a data base or shared file system). In some such systems the clients communicate directly with one another in a peer–peer architecture. However, peer–peer architectures often involve complex and hence error-prone coding, especially when one tries to account for participants joining and leaving in the midst of a collaborative interaction. A centralised notification server can help this by acting as a mediator between the clients.

For a particular update event one client will be the active client (AC) that performed the update and the rest will be the passive client (PC) who need to know that the update has happened (so they can update their display of the shared data). This leads to the architecture in figure 2.

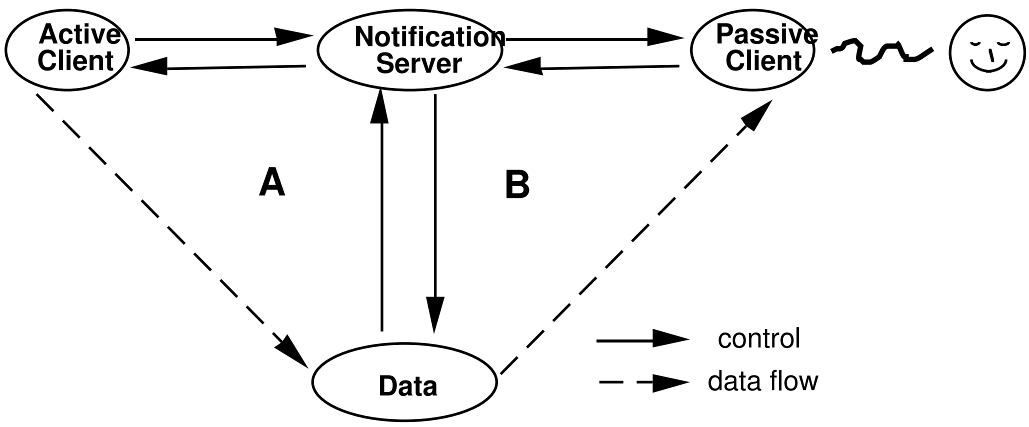

Figure 2. Notification server as mediator

The two sides of this, A and B, can be considered separately and each are of the form found in figure 3.

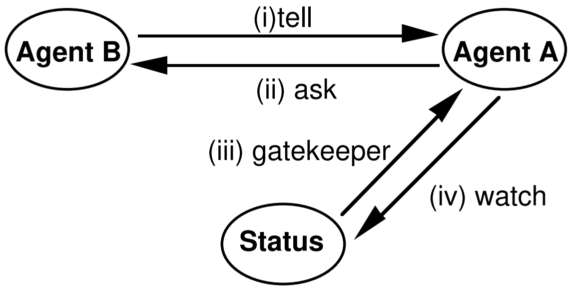

Figure 3. Notification server as mediator

This uses the 4 source/initiative alternatives for agent targets from section 4.4:

(i) A→A source agent **tells** target agent

(ii) A←A target agent **asks** source agent

(iii) S→A **gatekeeper** of status tells target agent

(iv) S←A target agent **watches** source status

When applied to the active client to notification server side (AC–NS), all four possibilities may occur. For example, the active client may tell the notification server when it updates the shared data– (i) tell. Alternatively, the notification server may poll the shared data – (iv) watch.

On the notification server to passive client side, we need not consider the event interactions between passive client and shared data as this would render the notification server redundant. (Although PC may go to the shared data for the value of the change.) Thus only options (i) and (ii) need be considered. Existing and potential notification server designs can then be placed the resulting 2x4 matrix.

Each position in this matrix has a corresponding chain description:

		notification server to passive client NS – PC (i) tells	(ii) asks
active client to notification server AC – NS	(i) tells	AC ↗ NS→PC ↘ SD	AC ↗ NS←PC ↘ SD
	(ii) asks	AC ↙ NS→PC ↘ SD	AC ↙ NS←PC ↘ SD
	(iii) gatekeeper	AC→SD→NS→PC	AC→SD→NS←PC
	(iv) watches	AC→SD←NS→PC	AC→SD←NS←PC

This analysis has enabled us to identify gaps where there are no current notification servers. For example, we could find no example which fits in slot (i)–(i) but this is clearly the 'purest' notification server. We are using this knowledge to guide practical work in notification server design and the construction of an experimental notification server called GtK (Getting-to-Know).

For a more detailed description of this analysis of notification server design see [39].

7. Related work

Notification and awareness

Shared information is of central importance in many collaborative systems, both explicit information such as shared documents and also implicit information on the presence and activities of other participants required to give a sense of mutual awareness [23, 30]. Because of this number of systems have been designed over recent years which act as 'notification' or 'awareness' servers including the Web Awareness Protocol [35], Lotus NTSP [38], Aether [42] and our own notification server GtK (Getting-to-Know) [39]. The principal purpose of these is not to store or manage data (although some do this as well), but to inform other programs and ultimately users when changes have occurred.

In addition, there has been considerable interest in formal models of awareness within collaborative applications [2-4, 40]. It is interesting to note that these formal awareness models are not phrased in event terms such as "when person A enters the room, person B is informed". Instead they are much more status-oriented: "when the nimbus (region of influence) of person A intersects that of person B they should be aware of one another".

Toolkits and programming

Status–status mappings are common in single user interfaces (e.g. dragging) and multi-user interfaces (e.g. keeping several users' views consistent). In addition, user interface events, such as mouse clicks, need to be passed on to the relevant component. Not surprisingly toolkits and user interface development systems often have some form of event notification mechanism.

The Smalltalk Model-View Controller (MVC) model used a mechanism whereby objects could register themselves as dependants of another object which would then inform its dependants about changes to its state [26, 31]. Similar techniques can be found in the X-Motif callbacks [34] and in the Java JDK 1.1 source–listener event model [25].

A similar mechanism is the use of active values, variables which invoke callbacks when modified. These were first used in the InterLisp environment within a single machine, but have also been adapted in Suite [10, 11] as the main mechanism for a distributed multi-user interface development toolkit.

A more sophisticated option is when the toolkit embodies a constraint maintenance system as in the case of Garnet [33] for single user interfaces and Rendezvous [28, 29] for distributed, multi-user interfaces.

In the callback-style toolkits the programmer must explicitly make the shift between thinking about relationships between status phenomena and code this in terms of event callbacks. In contrast, these constraint-based toolkits allow the programmer to think in terms of status–status relationships. They thus fall very close to the spirit of status–event analysis.

Formal notations

There has been a tendency for formal notations in computing in general and in HCI in particular to be event-oriented (or possibly event-fixated). This is natural given the discrete nature of digital machines. However, one of the purposes of these notations is to specify a user-oriented view of the system, which should not be limited by the discrete nature of current silicon technology. In previous papers, formal interface notations have been reviewed in relation to their adequacy for status–event description [1, 14], so only the most pertinent are described here.

Early formal interface notations based on process algebras or formal grammars were completely event-oriented. However, influenced by the PIE model [22], most current notations incorporate some form of mapping to capture the visual aspects of interface objects. The York interactor model [24] has a *render* mapping which makes parts of the interactor's internal state visible – that is they have status output, but not input. The CNUCE interactors [36, 37] also have such a rendering, but as they are specified in Lotos (a CCS derivative), they cannot express status outputs directly, but instead rely on specific 'get(x)' events, a form of polling.

Looking beyond user interface formalisms, we find that requirements specification has similar issues, for example Moffet *et al.* used an up-arrow notation (↑x) for 'predicate becomes true' events as part of their requirements capture model [32]. However, it is when formal methods meets physical systems that the need for status representation becomes unavoidable. In these circumstances discrete digital controllers need to interact with physical phenomena operating under the normal continuous processes of nature. This has lead to the study of so called 'hybrid systems' [27] with notations, for example the extended duration calculus [8, 9], which include differential equations for modelling physical phenomena and standard computing formalisms for the digital parts. The need to represent external phenomena as status has not lead to a recognition of the importance of internal status on an equal footing with event.

8. Conclusions and discussion

We have seen how status–event analysis can be used to systematically breakdown the steps in event discovery. The systematicity is important so that we do not omit cases during the analysis of specific problems. Of course, in specific domains, we may be able to ignore some of the generic alternatives as we saw in the analysis of the notification server design space, but it is important that we deliberately omit alternatives during design rather than accidentally miss them.

Although the key issue is to obtain a systematic and complete analysis, there are also some fascinating individual issues. The apparent reversal of initiative and 'causality' is one. Of course, at one level the causality and initiative must lie together. If agent A polls agent B to see whether event E has happened, there is a low-level flow of causality from A to B. However, at a higher level, periodic polling can be seen as simply an 'implementation issue' and we see the high-level event causality flowing from B to A.

This paper treads a middle ground between informal status–event timelines and full specifications. There was not space to delve into a full semantic model of status–event phenomena. This would help to elucidate some of these issues, but does need care if status phenomena are to be represented accurately.

We have seen how the causal chain analysis in this paper can help elucidate the software architecture issues for notification server design. However, the same causal chains can be seen in office procedures and other hum–human interactions. The power of status–event analysis is that it can account for human, physical and computational phenomena within a single conceptual framework.

Acknowledgements

This work was partly funded by EPSRC MNA grants GR/L64140 & GR/L64157, "Interfaces and infrastructure for mobile multimedia applications", and UK/HK Joint Research Scheme project 9050101, "Modelling Behaviour of Agents in CSCW".

References

1. Abowd, G. and A. Dix. *Integrating status and event phenomena in formal specifications of interactive systems*. in *SIGSOFT'94*. 1994. New Orleans: ACM Press. p. 44–52.
2. Benford, S., A. Bullock, C. Cook, P. Harvey, P. Ingram and O. Lee, *From rooms to cyberspace: models of interaction in large virtual computer spaces.* Interacting with Computers, 1993. **5**(2): p. 217–237.
3. Benford, S. and L. Fahlén. *A spatial model of interaction in large virtual environments*. in *Proceedings of ECSCW'93*. 1993. Kluwer Academic. p. 109–124.
4. Benford, S., L. Fahlen, C. Greenhalge and J. Bowers. *Managing mutual awareness in collaborative virtual environments*. in *Proceedings of ACM SIGCHI conference on Virtual Reality and Technology – VRST'94*. 1994. Singapore: ACM Press.
5. Brewster, S.A., *Providing a structured method for integrating non-speech audio into human-computer interfaces*. 1994, PhD Thesis, University of York, UK.
6. Brewster, S.A., P.C. Wright, A.J. Dix and A.D.N. Edwards. *The Sonic Enhancement of Graphical Buttons*. in *Human–Computer Interaction – Interact'95*. 1995. Lillehammer: . p. 43–48.
7. Brewster, S.A., P.C. Wright and A.D.N. Edwards. *The design and evaluation of an auditory-enhanced scrollbar*. in *Proceedings of CHI'94*. 1994. Boston, Massachusetts: ACM Press, Addison-Wesley. p. 173-179.
8. Chaochen, Z., C.A.R. Hoare and A.P. Ravn, *A calculus of durations.* Information Processing Letters, 1991. **40**(5): p. 269–276.
9. Chaochen, Z., A.P. Ravn and M.R. Hansen, *An extended duration calculus for hybrid real-time systems,* in *Hybrid Systems,* R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, Editors. 1993, LNCS 736, Springer-Verlag: p. 36–59.

10. Dewan, P. *A tour of the Suite user interface software*. in *UIST'90: Proceedings of the 3rd ACM SIGGRAPH Symposium on User Interface Software and Technology*. 1990. ACM Press. p. 57–65.
11. Dewan, P. and R. Choudhary, *A high-level, and flexible framework for implementing mulituser interfaces.* ACM Transaction on Information Systems, 1992. **10**(4): p. 345–380.
12. Dix, A., *An agent based architecture for groupware applications*, Editor^Editors. 1993, Computer Science Department, University of York:
13. Dix, A. and G. Abowd, *Delays and Temporal Incoherence Due to Mediated Status–Status Mappings.* SIGCHI Bullitin, 1996. **28**(2): p. 47–49.
14. Dix, A. and G. Abowd, *Modelling status and event behaviour of interactive systems.* Software Engineering Journal, 1996. **11**(6): p. 334–346.
15. Dix, A. and S.A. Brewster. *Causing Trouble with Buttons*. in *Ancillary Proceedings of HCI'94*. 1994. Glasgow: .
16. Dix, A., J. Finlay, G. Abowd and R. Beale, *Human–Computer Interaction (Second Edition)*. 1998, Prentice Hall.
17. Dix, A., D. Ramduny and J. Wilkinson, *Interaction in the Large.* Interacting with Computers - Special Issue on Temporal Aspects of Usability (to appear), 1998. .
18. Dix, A.J., *Formal Methods for Interactive Systems*. 1991, Academic Press.
19. Dix, A.J. *Status and events: static and dynamic properties of interactive systems*. in *Proceedings of the Eurographics Seminar: Formal Methods in Computer Graphics*. 1991. Marina di Carrara, Italy: .
20. Dix, A.J. *Beyond the interface*. in *Engineering for Human-Computer Interaction: Proceedings of IFIP TC2/WG2.7 Working Conference*. 1992. Ellivuori, Finland: North-Holland. p. 171-190.
21. Dix, A.J., D. Ramduny and J. Wilkinson. *Long-Term Interaction: Learning the 4 Rs*. in *CHI'96 Conference Companion*. 1996. Vancouver: ACM Press. p. 169–170.
22. Dix, A.J. and C. Runciman. *Abstract models of interactive systems*. in *People and Computers: Designing the Interface*. 1985. Cambridge University Press. p. 13-22.
23. Dourish, P. and V. Bellotti. *Awareness and coordination in shared workspaces*. in *CSCW'92*. 1992. Toronto, Canada: ACM Press. p. 107–113.
24. Duke, D.J. and M.D. Harrison, *Abstract Interaction Objects.* Computer Graphics Forum, 1993. **12**(3): p. 25–36.
25. Flanagan, D., *Java in a Nutshell 2nd Edition (Java 1.1)*. 1997, O'Reilly.
26. Goldberg, A., *Smalltalk-80, The interactive programming environment*. 1984, Addison-Wesley.
27. Grossman, R.L., A. Nerode, A.P. Ravn and H. Rischel, ed. *Hybrid Systems*. 1993, LNCS 736, Springer-Verlag: .
28. Hill, R.D. *The Rendezvous constraint management system*. in *UIST'93: Proceedings of the ACM Symposium on User Interface Software and Technology*. 1993. ACM Press. p. 225–234.
29. Hill, R.D., T. Brinck, S.L. Rohall, J.F. Patterson and W. Wilner, *The Rendezvous architecture and language for constructing multi-user applications.* ACM Transactions on Computer-Human Interaction, 1994. **1**(2): p. 81–125.
30. Lauwers, J.C. and K.A. Lantz. *Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems*. in *Proceedings of CHI'90 Human Factors in Computing Systems*. 1990. Seattle, Washington, April 1990: ACM Press. p. 303–311.
31. Lewis, *The Art and Science of Smalltalk*. 1995, Prentice Hall.
32. Moffet, J., J. Hall, A. Coombes and J. McDermid, *A model for a casual logic for requirements engineering.* Requirements Engineering, 1996. **1**(1): p. 27–46.
33. Myers, B.A., D.A. Guise, R.B. Dannenburg, B. Vander Zanden, D.S. Kosbie, E. Pervin, A. Mickish and P. Marchal, *Garnet: comprehansive support for graphical, highly interactive user interfaces.* IEEE Computer, 1990. **28**(11): p. 71-85.

34. OSF, *OSF/Motif Programmer's Guide, Revision 2*. 1995, Open Software Foundation, Prentice Hall.

35. Palfreyman, K. and T. Rodden. *A Protocol for User Awareness on the World Wide Web*. in *Proceedings of CSCW'96*. 1996. Boston, Massachusetts, Nov. 1996: ACM Press. p. 130–139.

36. Paternó, F. and G. Faconti. *On the use of LOTOS to describe graphical interaction*. in *Proceedings of HCI'92: People and Computers VII*. 1992. Cambridge University Press. p. 155–173.

37. Paternó, F., M.S. Sciacchitano and J. Lowgren. *A user interface evaluation mapping physical user actions to task-driven formal specifications*. in *Design, Specification and Verification of Interactive Systems '95*. 1995. Springer Verlag. p. 155–173.

38. Patterson, J.F., M. Day and J. Kucan. *Notification servers for synchronous groupware*. in *Proceedings of CSCW'96*. 1996. Cambridge, Massachusetts: ACM Press. p. 122–129.

39. Ramduny, D., A. Dix and T. Rodden. *Getting to Know: the design space for notification servers*. in *Proceedings of CSCW'98*. 1998. Seattle, Washington: (to appear).

40. Rodden, T. *Populating the application: a model of awareness for cooperative applications*. in *Proceedings of CSCW'96*. 1996. Boston, Massachusetts, Nov. 1996: ACM Press. p. 87–96.

41. Rodden, T., K. Cheverst, N. Davies and A. Dix. *Exploiting Context in HCI design for Mobile Systems*. in *Workshop on Human Computer Interaction with Mobile Devices*. 1998. Glasgow, May 1998: .

42. Sandor, O., C. Bogdan and J. Bowers. *Aether: an awareness engine for CSCW*. in *Proceedings of ECSCW'97*. 1997. Lancaster: Kluwer Academic. p. 221–236.

43. Wood, A., *CAMEO: Supporting Agent-Application Interaction*. 1998, PhD Thesis, University of Birmingham, UK.

44. Wood, A., A.K. Dey and G.D. Abowd. *CyberDesk: Automated Integration of Desktop and Network Services*. in *Proceedings of the 1997 conference on Human Factors in Computing Systems, CHI '97*. 1997. ACM Press. p. 552–553.