INTERACTIVE SITUATION MODELS FOR INFORMATION SYSTEMS DEVELOPMENT

Meurig Beynon, Richard Cartwright, Pi-Hwa Sun, Ashley Ward
Department of Computer Science, University of Warwick, Coventry CV4 7AL

ABSTRACT

An interactive situation model (ISM) is a new kind of artefact to support information systems development. An ISM is particularly well-suited for use in the early stages of the development process. Computer-based ISMs can be constructed using Empirical Modelling (EM) principles and software tools that have been developed at Warwick. The developer construes system behaviour in terms of three key fundamental concepts of EM: observables, dependencies and agents. Observation-oriented analysis is used to formulate an explanatory account of system using a novel notation called LSD. A special-purpose software tool, such as the EDEN interpreter, is used to develop an associated ISM. In the environment supplied by this ISM, the developer can explore the implications of different contexts and scenarios for interaction between agents.

This paper outlines the principles of EM, and illustrates the ISM concept with reference to a digital watch and statechart simulation. The use of ISMs as a means of shaping state and agent interaction is compared and contrasted with the use of statecharts. Recently published work by Horrocks on the application of statecharts to the design of user-interface software provides the context for this study. A refinement of Horrocks's event-state-action paradigm for user-interface specification and implementation is proposed.

Keywords

software development, statechart, interactive artefacts, user-interface design, situated modelling, event-driven, agent-oriented

1. INTRODUCTION

The convergence of software development methodologies in recent years has given prominence to artefacts that can support the development process. In the Unified Modelling Language (UML) that has emerged, there are many standard diagrammatic techniques that can be used to describe relevant aspects of the system being developed. Whilst there is some computer support for constructing these artefacts, principally in the form of graphical user interfaces, this is oriented towards a static view of artefacts that belies their role in the dynamic and volatile processes associated with the early stages of system development.

The complexity of the interactions between programmable components and human agents in modern computing applications motivates new abstractions for the conceptual representation of information systems. Understanding the role of human and inanimate components within a reactive system, for instance, involves not only input-output transformations, but also communication and stimulus-response issues.

This paper proposes novel computer-based interactive situation models (ISMs) to assist information systems development. An ISM provides an environment within which the human interpreter can explore the relationships between observables and the patterns of behaviour associated with a system component, with particular reference to its external real-world semantics. Section 2 introduces the ISM concept. Section 3 describes the principles and tools that we have developed for constructing ISMs. The way in which our approach is related to conventional methods of state representation, such as the use of statecharts, is illustrated by an ISM we have devised for a digital watch. Section 4 develops this theme with particular reference to the principles for user-interface design described by Horrocks in [5]. Key ideas of the paper are elaborated in [1], from which the content of section 3 has been drawn.

2. THE ISM CONCEPT

In modern software engineering, the emphasis has shifted from computer programming to systems modelling and development. The most significant and difficult aspects of the software engineering process precede the explicit specification and generation of code. Artefacts, such as those introduced in UML, play a central role. The potential benefit of computer support for creating and communicating such artefacts is clear, but this needs to be more than superficial and syntactic. ISMs are intended to complement the traditional artefacts used in systems development, and to address the creative conceptual activities that accompany their construction.

A computer-based artefact is very different in character from a computer program. Whereas a program fulfils a preconceived and specified function, an artefact can serve as an open-ended aid to conception and design. An artefact is specifically intended for human contemplation and interpretation. Though there are conventions about how certain artefacts are to be interpreted, it is of the essence that the construction of artefacts accompanies and informs the process of gaining and communicating insight. Our experience of observing and interacting with artefacts shapes our conception of the systems we seek to identify, analyse or develop.

Though the artefacts used in current practice are predominantly static text documents and diagrams, change features prominently in their use. A typical artefact is intimately linked to its situation in the natural and social world. The interaction and negotiation that establishes this link shapes both the artefact and its interpretation. The interactive situation model arises from exploiting computer technology to create artefacts that can be more conveniently and effectively modified and reinterpreted.

Our interactive situation models are constructed using principles and tools that have been developed in the Empirical Modelling (EM) project here at Warwick. Constructing ISMs to represent the interactions between the agents in a reactive system has been a central focus for research in EM. In this context, an *agent* refers broadly to any component of the system that can be responsible for changing state: for example, a computer, a sensory device, a switch, an actuator, a clock or a human agent. The digital watch and statechart simulation in Figure 1 is one example of such an ISM (cf. [2]). Other examples described elsewhere include a vehicle cruise controller, a billiards game simulation, and a railway simulation. For details, see the references in [1] and the EM web site at `http://www.dcs.warwick.ac.uk/modelling/`.

The key idea behind the ISM concept is that of representing the way in which the modeller construes a situation. The best precedents for constructing ISMs are to be found in experimental science, where scientists typically communicate their experience of unfamiliar phenomena with reference to interaction with artefacts devised using familiar objects and abstractions. In analysing Faraday's experimental practices, Gooding [3] introduces the term 'construal' to refer to artefacts of this kind. His characterisation can be applied both to the artefacts of UML and to ISMs: "Construals are a means of interpreting unfamiliar experience and communicating one's trial interpretations. Construals are practical, situational and often concrete. They belong to the pre-verbal context of ostensive practices." ([3], p.22); "... a construal cannot be grasped independently of the exploratory behaviour that produces it or the ostensive practices whereby an observer tries to convey it." ([3] p.88).

Figure 1 depicts an ISM of a digital watch such as might be constructed by a proficient user. It has three components. Figure 1.1 represents the significant features of the watch, providing buttons that imitate its standard functions. Figure 1.2 reveals how the time displayed on the digital watch is to be interpreted. Figure 1.3 captures the expectations of the user about the possible states and transitions of the watch in response to button input. The status of Figure 1 as a construal, in the sense of Gooding, relies upon the potential openness of the ISM to interaction: the modeller can not only simulate standard interactions with the watch, and observe the way in which the states of the three components are consistently transformed, but can also experiment in ways beyond the scope of mere watch simulation. For instance, the modeller can: define the time arbitrarily; simulate time reversal; refine Figure 1.1 to better resemble the actual appearance of a watch; adjust the statechart by revising states and labels or adding annotations. Most significantly, such modifications can be performed seamlessly in one and the same interactive environment, using a single paradigm for interaction, without the need of elaborate processes

of redesign and re-compilation. The application of EM principles plays an essential role in this.

3. EM PRINCIPLES FOR ISMS

EM supplies a framework for construing concurrent systems in which the basic abstractions are observables, dependencies between observables, and agents that act through changing observables and dependencies. Simple informal definitions for these concepts are:

- an **observable** is some feature of a system to which a value or status can be attributed in a system state. Empirical procedures and conventions are involved in identifying a particular observable and assigning its value. Not all the observables associated with a system need be present in a particular system state.
- an **agent** is a family of observables whose presence and absence in system states is correlated in time, that is typically deemed to be responsible for particular changes to observables within the system. All changes to the values of observables within a system are typically construed as due to actions on the part of agents.
- a **dependency** is a relationship between observables that pertains in the view of a particular agent. It expresses the empirically established fact that when the value of a particular observable x is changed, other observables (the dependants of x) are of necessity changed in a predictable manner as if in one and the same action. The changes to the values of x and its dependants are indivisible in the view of the agent. That is: no action or observation on the part of the agent can take place in a context in which x has changed, but the dependants of x have yet to be changed.

The identification of observables, dependencies and agents is arguably an activity that is implicit in all system construction, whatever development method or programming paradigm is used. In applying EM principles, the quality of the developer's construal determines the extent to which an ISM can be easily modified and re-interpreted.

In EM, construing a situation involves two complementary activities:

- describing the abstract explanatory account of the situation in the modeller's mind;
- constructing an ISM to imitate the observed responses to experimental and exploratory interaction.

Describing the explanatory account involves agent-oriented analysis based on a special-purpose notation called LSD. This analysis generates a text document: an *LSD account* of the situation. The LSD account captures the modeller's conception of the situation with reference to what observables and dependencies are significant, what agents are present, and how observables are classified with respect to these agents.

Constructing an ISM involves creating an artefact to realise a pattern of observables, dependencies and agency that faithfully reflects observation of its referent. We have developed several software tools for this purpose. The one most extensively used, whose use is illustrated by the ISM in Figure 1, is the EDEN interpreter.

Brief details of LSD and EDEN will be explained with reference to Figure 1.

3.1. An LSD account of the Digital Watch

Listing 1 is an outline of the LSD account for the digital watch that is associated with the ISM in Figure 1. This account is deliberately modelled on the analysis of a digital watch that informs Harel's statechart (cf [2,4]). It can be read as expressing how the modeller conceives the operation of the watch. The physical watch is represented in each state by instantiations of agents specified in the LSD account. The watch agent can be regarded as representing the watch, and the power agent its power supply.

Within the specification of each agent the identifiers refer to observables that are significant for the agent. Certain observables are associated with the agent itself, and do not exist when the agent is absent. Such an observable is classified as a **state** for an agent. For instance, `power_s` represents the voltage that is supplied to the watch: this is not a meaningful observable in the absence of the battery, and can otherwise take on three values, according to whether its charge is strong, weak or zero. Within an agent, the special state observable `LIVE` indicates whether the agent is present or absent. A **derivate** for an agent declares a dependency between observables in the view of that agent, so that for example:

```
LIVE = (power_s >= 1)
```

indicates that the watch is operative whilst the power is not zero.

The specifications of the watch and main agents include subagents. The pattern of instantiation of these subagents determines how they contribute to the functionality of the agent in which they appear. In the case of the watch agent, the subagents are instantiated whilst it is instantiated, as is explicitly indicated by the LIVE derivates for the subagents `main` and `alarm_st`. In the case of the `displays` agent, precisely one of the subagents is instantiated whilst `displays` is active, according to which mode of display (as determined by the values of the observable `displays_s`) currently pertains. These two ways of composing a family of subagents are closely related to the concepts of *orthogonality* and *depth* in a statechart (cf. [3]).

The behaviour of agents within a system is construed as determined by a protocol for action. Each action in the protocol is expressed by a possible stimulus-response pattern. For example, for the `alarm_st` agent, the interpretation of the action

```
displays_s = A & alarm_s = D & !d
                → alarm_s = E
```

is: when the alarm time is being displayed and the alarm is currently disabled, a possible response to pressing button `d` is to enable the alarm. An observable that is deemed to influence the behaviour of an agent (such as `displays_s` or `alarm_s` in this context) is classified as an **oracle**. An observable that can be redefined by an agent in the course of an action is classified as a **handle**.

The LSD account is not to be mistaken for a formal specification of system behaviour. It is concerned only with how state-changing actions are attributed to agents, and how their interaction is mediated through observables at their interfaces. The context for agent interaction, and the viewpoint of an objective external observer are conspicuously absent. In framing an explanatory account and constructing an ISM, the modeller adopts an empirical stance. The LSD account reflects what the modeller construes to be causal connections between changes to system state on the basis of empirical evidence. The ISM serves as a metaphorical representation for the system, consistent with the modeller's construal. Interaction with the ISM then enables the modeller to assess the quality of their explanatory account by exploring the implications of particular patterns of agency and interaction.

3.2. Constructing an ISM

The scope for modifying and re-interpreting ISMs stems from the way in which states and transitions are represented in EM. The key idea is that the ISM is a model of the relationship between a situation and an observer, so that a change can either reflect a development in the situation ("an external change of state") or a new realisation on the part of the observer ("an internal change of mind"). The character of an ISM in this respect resembles a spreadsheet, where the semantics of possible actions that might be performed by the user include: the update of a cell to reflect a change in the external situation; the correction of a cell value to make it consistent with the current situation; the modification of a defining formula to reflect a new insight into the situation, either to correct or to refine; the introduction of a new cell and defining formula to record additional information about the situation.

In an ISM, the situation is represented by a script of definitions (a *definitive script*) resembling the defining formulae in a spreadsheet. (The ISM in Figure 1 contains several hundred definitions.) In this context, the term *definition* refers to an expression of the form $q = f(x, y, z, \ldots, t)$, where $q, x, y, z, \ldots, t$ are variables in the script. Each variable represents an observable in the situation, and each definition a dependency between observables. The current values of variables in the script should conform to the current observations made of the corresponding observables in the situation. These values then metaphorically represent the current situation. Transitions within the ISM are associated with redefinitions of variables (possibly performed in parallel). Changes to values of variables are propagated in an indivisible fashion via the dependencies associated with definitions.

The EDEN interpreter maintains dependencies in definitive scripts in which variables have many different types. An important feature of EDEN is that scripts can be formulated using variables whose values are geometric in nature: these include points, lines, and windows. This is illustrated in Figure 1, where the digits of the LCD display and the positions of the hands of the analogue clock are functionally dependent on the current time, as

recorded internally as a scalar value (viz. "the number of seconds elapsed since a reference time").

In appreciating interactive use of the ISM, it is helpful to imagine an idealised state-changing regime in which the modeller has discretionary control over every redefinition that occurs. Conceptually, the changes of state in the ISM should reflect as far as possible the expectations of the modeller. The dependencies in the model should account for all changes that are indivisible and cannot be interrupted (at least from the modeller's perspective). All other changes of state observed in the situation should be consistent with the modeller's explanatory account of the system behaviour, in the sense that they are plausibly attributable to the actions of agents represented in the LSD account. The state of the ISM can then be updated by the modeller, who performs the appropriate family of parallel redefinitions.

4. ISMS AND USER-INTERFACE DEVELOPMENT

In [5], Horrocks identifies the need for an alternative to the bottom-up "event-action" paradigm for the design of user-interface (UI) software. He proposes a three layer UI-Control-Model architecture in which the control layer is concerned with invoking appropriate actions in response to events, maintaining current state and ensuring the coherence of UI objects. The purpose of the control layer is to make explicit the state information that is implicitly introduced when a UI is constructed bottom-up. The state ingredient to which Horrocks refers is identified with the set of currently enabled events. Horrocks advocates the use of statecharts to specify these sets, pointing out that orthodox finite state machine models are far too complex to serve as useful artefacts in general UI design.

Horrocks's perspective as a practitioner of UI software design has both abstract and technical relevance to our theme. His central concern is not with the use of statecharts *per se*, but with the processes by which statecharts can be developed. In particular, he remarks that "there is very little written about how to determine the contexts in which events occur" ([5] p.202). He also identifies significant obstacles to applying traditional techniques to UI software design - citing the limitations of implementing control through global variables and procedures, and the difficulties of applying object-oriented principles where there is a need to access non-global variables and to maintain state that is distributed over many UI objects. A particular concern is that of "writing UI code that can be repeatedly enhanced" ([5] p.207).

In the EM framework, *the modeller as agent* is the archetype for all agents within the system. In its agent-oriented system development, specifying the interfaces between agents is a key problem, for which specifying a UI is the archetypal counterpart. From an EM perspective, the events that are enabled in a particular context are determined by the agents and observables that are present, and by the privileges these agents currently have. The experimental activities associated with EM are precisely directed at determining which events can occur in a particular context.

As our interpretation of the statechart for the digital watch indicates, each state of the statechart signifies the presence of an agent in a particular role. Where Horrocks proposes a paradigm shift from event-action to event-state-action, EM may be viewed as commending an event-state-*agent*-action approach. Our experience of EM has shown its distinctive merits as a new approach to UI software design. Agent-oriented analysis combined with dependency maintenance serves to rationalise the side-effects of procedures that operate on global variables; though violating object-oriented principles, it can guarantee the integrity of state that is distributed across UI objects.

Horrocks's approach to UI software design can be examined from an EM perspective. His prescription for a statechart model for the control layer ideally presumes that an agent's influence is characterised by its presence alone, and that its privileges are independent of the state of concurrently acting agents ([5] p.101). The statechart that Horrocks uses to specify the UI to a calculator reflects such a model: the agents associated with the statechart are equipped for specific lexical processing tasks (such as reading a character or digit) and are invoked to a precise pattern dictated by an elementary parser for arithmetic expressions. Such a highly constrained context for user-interaction may be atypical, but it is nevertheless clear that a statechart is appropriate where the interplay between agent roles and their contexts for their adoption have been comprehensively determined.

A significant issue for Horrocks's approach is the distinction between what in EM terms might be described as *genuine* agents (which have some state-changing capability) and *object-like* agents (which have no state-changing protocol). The agents to be

represented in his statecharts are genuine agents that are deemed to be part of the UI. These distinctions are illustrated by the ISM for the digital watch. Some aspects of the model (such as determine the role of the digits that appear on the display) can be deemed not subject to change. Other aspects describe internal state-changing agents (for example, to reset the time) that can be seen as separate from the user-interface. Reference to observables associated with the internal model of the watch in statechart transitions is legitimised by respecting these distinctions.

Distinguishing the UI from the model also raises significant issues. For instance, the Visual Basic calculator that is used to motivate Horrocks's calculator case study ([5] p.20) has a latent complex agency, whereby the input generated by a digit button is determined by its caption. This feature could be usefully exploited to reconfigure the interface in a calculator that allowed (e.g.) a change of number base to hexadecimal. User-interface models that support such subtle synergy between the UI and the underlying model may also be required in other contexts, for example, where the goal is to diagnose faults, or to design for fault tolerance.

Context-dependence in the UI interface that involves observables attached to genuine agents may be problematic in the development of a user-interface using Horrocks's approach. By way of illustration, if many users of a shared water tank have independent concurrent access to `fill_tank` agents, powerful use can be made of context-dependent privileges. For instance, an agent that fills by incremental steps, and can act only when the tank is not full, has several useful properties: it stops when the tank is full; if appropriately synchronised, it can also operate in parallel with a second `fill_tank` agent. Provided that the tank itself is treated as an object that has no state-changing protocol to be represented in the statechart for the UI, Horrocks's design heuristics can be respected. It may yet be hard to revise the statechart if the tank is subsequently regarded as subject to independent state-changes: for instance, if the effects of expansion, evaporation and leakage are to be taken into account.

There are good prospects for addressing issues of this nature using ISMs. The most significant feature of their use is that their semantics is moulded through situated interaction, rather than abstract and preconceived. Issues such as whether an agent is object-like or whether it has a role in the UI are always at the modeller's discretion, and in principle remain open at every stage of the development. Environments for exploring and framing the contexts in which events occur demand a model of user interaction beyond the scope of the statechart approach. Our experience shows that definitive scripts support state representations that surpass the statechart in expressive power. They are particularly effective in shaping the contexts that mediate stimuli and responses. The openness of the UI in EM can be exploited in debugging and testing, with potential for flexible redesign and fault-tolerance.

CONCLUSION

The use of ISMs for UI software development is one of several potential applications of ISMs to issues in systems development that we are currently investigating. Related research has addressed requirements understanding, software construction, concurrent engineering and program comprehension (see [1]). An important feature of our approach is that traditional artefacts, such as the statechart in Figure 1, can be modelled using EM principles and seamlessly integrated into an ISM. This motivates the potential use of EM principles and tools to provide interactive support for traditional methods and artefacts for systems design. A more radical alternative, the automatic generation of conventional programs from ISMs, is also being explored.

ACKNOWLEDGEMENTS

We thank members of the Empirical Modelling Research Group for contributions to this paper.

REFERENCES

[1] W M Beynon, R I Cartwright, J Rungrattanaubol, P-H Sun, *Interactive Situation Models for Systems Development*, CS RR#353, University of Warwick, 1998.

[2] W M Beynon, R I Cartwright, *Empirical Modelling for Cognitive Artefacts*, IEE Colloquium, Dec 1995, Digest #95/231, 8/1-8/8.

[3] David Gooding, *Experiment and the Making of Meaning*, Kluwer Acad Pubs, 1990.

[4] D Harel, *Statecharts: a visual formalism for complex systems*, Science of Computer Programming, 8:323-364, 1987.

[5] I Horrocks, *Constructing the User Interface with Statecharts*, Addison-Wesley, 1999.

Listing 1: Outline LSD account for the Digital Watch (cf. Figure 1)

```
agent power() { state power_s = battery_charge }                                    // three-valued 2,1,0
agent watch() {
    derivate   LIVE = (power_s >= 1)
    oracle     power_s
    agent main() {
        derivate   LIVE = LIVE_watch
        oracle     LIVE_watch , main_s, alarm_s
        state      main_s = D                                   // an integer -- 1:Displays, 2:Beep || displays
        handle     main_s
        protocol   (main_s == D) & (time == set_time) & alarm_s == E → main_s = B
        agent displays() {
            derivate   LIVE = LIVE_main
            oracle     LIVE_main
            state      displays_s = T   // 1:Time, 2:Update, 3:Date, 4:Stopwatch, 5:Alarm, 6:Update Alarm, 7:Chime
            handle     displays_s, update_s, upalarm_s
            protocol   displays_s == T & !c → update_s = 1,          // !c here means that button c has been pressed
                       displays_s == T & !d → displays_s = D,
                       displays_s == A & !c → upalarm_s = 1,
                       displays_s ≠ S & displays_s ≠ T & 2-min → displays_s = T          // display timeout
                       ...
            agent disp_date() {
                derivate   LIVE = LIVE_displays & displays_s == D,
                           "watch_display = date as of clock()"
                oracle     LIVE_displays, displays_s
            }
            agent disp_time() { ... }
            agent disp_upalarm() {
                derivate   LIVE = LIVE_displays & upalarm_s > 0,
                           displays_s = (upalarm_s % 4 == 0 ) ? A : UA
                           "watch_display = time as of alarm setting with right digit highlighted"
                oracle     LIVE_displays, upalarm_s
                state      upalarm_s = M                  // 1:Min, 2:TenMin, 3:Hr, 4 ≡ 0(mod 4):Alarm
                handle     upalarm_s, set_time
                protocol   !b or 2-min → upalarm_s = 0,
                           !c → upalarm++,
                           "event→ update set_time so as to increment highlighted digit"
            }
            ...
        }                                                                    // end displays()
        agent beep() {
            derivate   LIVE = LIVE_main & main_s == B
            state      main_s
            protocol   beep_stop → main_s
        }
    }                                                                        // end main()
    agent alarm_st() {
        derivate   LIVE = LIVE_main
        oracle     LIVE_main, displays_s, alarm_s
        state      alarm_s = D, set_time = 00.00
        handle     alarm_s                                                   // 1:Disab, 2:Enab
        protocol   displays_s == A & alarm_s == D & !d → alarm_s = E
                   displays_s == A & alarm_s == E & !d → alarm_s = D
    }
    agent chime_st() { ... }
    agent clock() { ... }
    agent stopwatch() { ... }                          // ... and several other agents, such as light() etc...
}
```

Figure 1: The digital watch Interactive Situation Model