Empirical Modelling Principles for Cognitive Artefacts

Meurig Beynon and Richard Cartwright

Department of Computer Science, University of Warwick, Coventry CV4 7AL

Introduction

What is meant by a cognitive artefact? In this paper, we shall adopt the following interpretation. A cognitive artefact is an object or environment that is contrived for interaction that imitates interaction with some other (typically real-world) system. This means in particular that an artefact exhibits different states each of which corresponds to a state of the system it represents. Such a mode of representation can offer the possibility of open-ended interaction, allowing the human interpreter to explore the correspondence between the artefact and its referent beyond its preconceived limits. Its open-ended experiential nature distinguishes communication using artefacts from communication via documents expressed in a formal language, where the interpretation presumes a preconceived framework, and does not admit further elaboration.

The use of artefacts is well-established as a means of communication in many disciplines. Contemporary interest in artefacts has been greatly enhanced by the new possibilities associated with computer-centred technologies. Computers in particular offer unprecedented potential for representing state and animating changes of state, especially when used in conjunction with multimedia interface devices. In understanding the principles that underlie the use of artefacts in systems modelling, it is helpful to consider all manner of artefacts, whether they are computer-based or not. These include mechanical models of engineering systems, architectural models, molecular models, maps and globes etc., and encompass artefacts in which the only scope for state change is associated with a change in viewpoint on the part of the interpreter.

Cognitive artefacts typically serve a useful function in the design process in two rather different ways. There are artefacts with which the designer can interact in order to develop insight into the nature and current status of the object that is being designed. There are also artefacts which inform the designer about the current status and progress of the design process. Both kinds of representation are especially significant in relation to modern trends towards concurrent engineering (cf [1]) and an emphasis on cognitive aspects of software development [5]. Artefacts of several different kinds are necessary to take account of the interaction between the object and a whole range of potential design participants, users and scenarios.

2. Empirical Modelling for Cognitive Artefacts

The construction of computer-based cognitive artefacts is a central theme in the Empirical Modelling Project that is being carried out at the University of Warwick. In this project, we have developed principles for constructing such artefacts. These principles and their implications will be described and illustrated informally with reference to familiar artefacts for telling the time. For further details and other applications, see [1] and the references therein.

2.1. Refining the Concept of Cognitive Artefact

In our modelling method, the concept of correspondence between artefact and referent introduced above is made more explicit. Each state of the referent is specified in terms of the current values of a collection of observables. Primitive interactions with the referent, as defined by changes in viewpoint or manipulation of observables, are specified as synchronised changes to these observables. The artefact also has its own collection of observables and primitive interactions. The observables and primitive interactions of artefact and referent can be put into correspondence so that the patterns of simultaneous change of observables associated with primitive interactions with artefact and referent coincide. In effect, the current state of the referent and the potential primitive interactions

in that state, as observed from some viewpoint, are metaphorically reflected in the artefact.

The term *observable* is used in this context in a broad sense to encompass any discernible feature of a phenomenon that can be ascribed a value by a well-defined experimental procedure or convention for interpretation. With such a broad interpretation, it becomes possible to develop artefacts to represent complex systems that include both human agents and engineering devices. For instance, an artefact to represent the status of a concurrent engineering process may comprise information about what parameters particular designers are entitled to modify as well as views that incorporate specialised information about the current values of parameters.

Our more precise characterisation of a cognitive artefact has some advantages:

- it suggests a criterion for assessing the quality of an artefact;
- it caters for empirical knowledge about potential states and transitions;
- it allows composition and extension on-the-fly to reflect new insights.

2.2. Artefacts, Viewpoints and Agency

The states and observables of artefact and referent invoked in our characterisation of an artefact are associated with agency. The very concept of state presumes a viewpoint (who makes the observations?) and a snapshot (when are the observations being made?). A viewpoint is associated with an agent who can in general both observe and experiment with artefact and referent (if sometimes only through thought experiment). The role of such an agent is characterised by its personal observables, perceived relationships between observables and protocol for interaction. In our approach, these are the constituents of an agent specification in our special-purpose notation LSD.

The essence of an artefact is that it should imitate interaction with its referent. This presumes a metaphor for representing the current state of the referent, whereby observables of the computer model correspond to observables of the referent. Representing time by the positions of the hands on a clock or by a numerical display illustrates such a use of metaphor. At present, our empirical modelling tools enable us to exploit metaphors based on configuring windows that contain line-drawings or text and can be sensitive to mouse actions. Artefacts of this kind are implemented in the definitive (definition-based) notations Scout (for window layout) and DoNaLD (for line-drawing) using the EDEN interpreter (an Evaluator for Definitive Notations).

Interaction with the referent discloses characteristic patterns of change in observables that express how a change to the value of one observable in general involves a simultaneous functionally dependent change in the values of other observables. Each agent who interacts with the referent potentially experiences it through a different set of observables, and different dependency relationships between observables. In general, not all the changes to observables encountered in interaction with the referent can be directly attributed to the actions of the experimenter. In such cases, an appropriate artefact must represent the current state of a system of interacting agents.

2.3. Artefacts and the Faithful Representation of State

Faithful representation of the state of the referent in an artefact demands faithful representation of both observables and dependency relationships within a multi-agent model. The exact and explicit correspondence between states and transitions in artefact and referent is a fundamental goal of our empirical modelling method. Without such a correspondence, the whole concept of being able to recognise the association between an artefact and its referent is suspect. Experimentation with an artefact is what reveals the characteristic relationships between observables that determine the identity of the referent. An artefact is bound to its referent by this means.

Fastidious representation of state is accordingly a feature of our approach. For instance, whereas in a traditional program the state of the screen is determined by the side-effects of procedural actions, in our models the current state of the screen is treated as a part of

the computational state, and has its value explicitly defined by a system of variables whose values are appropriately synchronised in change to reflect the dependencies between graphical elements. This synchronisation is achieved by introducing into the computational model an implicit updating process that maintains dependencies between variables in a spreadsheet-like manner. The dependencies between variables that are to be maintained in this way are explicitly expressed in a script of definitions (or definitive script) similar in nature to the family of formulae that relate spreadsheet cells.

Definitive scripts serve to express the precise scope of the immediate effect of an agent action. By redefining one observable explicitly, an agent typically makes implicit changes to the values of several in an indivisible fashion. Where an artefact represents many interacting agents, simultaneous redefinitions serve to represent concurrent actions. Identifying the agents within the referent and the observables to which they respond and which they can redefine is an empirical process. Models of concurrent systems are constructed by first developing LSD specifications for these agents, then animating their interaction in the computational framework of the Abstract Definitive Machine (ADM). This mode of animation allows intervention by the modeller in the role of a superagent that has an unrestricted protocol for redefinition and can arbitrate where there is conflict. This means in particular that the pattern of interaction between agents modelled by the artefact is entirely at the discretion of the superagent. For this reason, it is possible to represent the effect of spontaneous interaction of human agents in a system, continuous processes and instantaneous events.

3. An Illustrative Example: Modelling a Timepiece

Figure 1 depicts artefacts to illustrate the main characteristics of our Empirical Modelling approach. There are three artefacts: models of a digital watch and an analogue clock, and a statechart that represents the functionality of the digital watch display mechanism (from Harel [2]). The choice of familiar artefacts is deliberate, as it simplifies our account of principles, methods and tools.

3.1. Agency and Dependency

Figure 1 illustrates two quite different ways in which the watch and clock models can be interpreted as artefacts. They can be regarded as timepieces, and evaluated as methods of communicating the time. For instance, the buttons on the digital watch might be linked to a mechanical interface that allowed the user to determine how conveniently they are laid out for manipulation. From another perspective, an artefact might serve as a model of the watch or the clock as an engineering device. For instance, the statechart serves to tell the designer how the display functions of the digital watch are affected by button presses.

Simple applications of definitions and agency are illustrated by the analogue clock in Figure 1. In an engineering model, the hands of the clock may be coupled mechanically so that when the minute hand moves, the hour hand moves simultaneously. This dependency can be expressed by an explicit definition (where angle_min_hand is measured modulo 24π) :

$$\text{angle\_hour\_hand} = (\text{angle\_min\_hand} / 2\pi) * (\pi / 6)$$

Whilst such a definition is extant in the model, any movement of the minute hand entails simultaneous movement of the hour hand, both in simulating the clock in its normal operation and in setting the clock. The normal operation of the clock is associated with agents that can act autonomously without the intervention of the designer. The designer can nonetheless intervene to interfere with these agents, for instance to simulate a clock running down or malfunctioning. The role of the user in setting the clock, which conforms to no preconceived pattern of interaction, can be played directly by the designer.

The relationship between second and minute hands illustrates another role for agency in constructing an artefact. In our chosen clock mechanism, the motions of the second and minute hands are mostly independent but are synchronised by updating the position of the minute hand at regular intervals. In this case, movement of the minute hand is not within the scope of the indivisible action of moving the second hand. To model this, we

introduce an independent agent that monitors the position of the second hand and updates the minute hand in discrete steps as the second hand registers that a minute has elapsed.

3.2. Statecharts as Cognitive Artefacts

In characterising artefacts, our emphasis on state and uncircumscribed interaction clearly distinguishes modelling with artefacts from traditional formal modelling. It is interesting that Harel, whose vision of software development for reactive systems [3] stresses models with formal semantics, should also emphasise the role of visual formalisms [4], and the use of statecharts in particular. The value of statecharts has been confirmed by their widespread adoption in modern software development methods (cf [6]). This is a motivation for interpreting statecharts as cognitive artefacts within our framework.

As explained in [4], two fundamental concepts behind the statechart abstraction are *depth* and *orthogonality*. These can be informally illustrated with reference to Figure 1.3. The *displays* state has depth because it entails being in one of several substates, viz: displaying the time, the date, the stopwatch, the alarm time etc. The *alive* state is a product of the orthogonal substates defined by the functions associated with the display, power, light, alarm, chime components. To interpret a statechart in Empirical Modelling terms, we read the states as specifying conditions for the liveness of agents. Where there is depth, we think of an agent playing more and more specific roles. Where there is orthogonality, we think of agents operating in parallel.

In this way, a statechart serves to represent the pattern of temporal coincidence between agents, subject to certain constraints being met. Where one agent is defined by several acting in parallel, they must share a common lifetime. Where one agent is to be regarded as a role for another agent, its lifetime must be nested within the lifetime of that agent. In Figure 1.3, these constraints indicate minor anomalies in the treatment of certain states. For instance, the beep() agent associated with the statechart is best conceived as having a lifetime briefer than the watch. When the circumstances that lead to beeping arise, the main() agent, responsible for display functions, runs a beeping process in parallel, until such time as the beeper is disabled. Modulo the rationalisation of such anomalies, the statechart is an artefact that corresponds closely to the LSD specification in Listing 1, from which our simulation of the digital watch was developed. This exercise incidentally shows the potential for statecharts as artefacts for representing patterns of interaction between designers in concurrent engineering. We can interpret a highlighted region of the statechart as representing a current mode of interaction with the digital watch.

3.3. The Empirical Modelling Process

The association between artefact and referent described above is of its essence empirical. Characteristic patterns of change in observables are identified through observation and experiment, and are always subject to revision in the light of subsequent observation. The process of developing an artefact naturally involves incremental construction that is guided by evolving knowledge about the properties of the referent. In this process, it is essential to be able to record the current status of partial models, and revise these to take account of new information and perspectives on the referent as they are encountered.

Empirical Modelling is well-suited to meeting these requirements, though this is not evident from inspection of the relatively simple artefacts in Figure 1. The power of our method is best appreciated by experiencing the development process rather than viewing snapshots of its products. Figure 2 is a screenshot that gives an impression of our development environment. It includes an input window in which EDEN fragments and DoNaLD or Scout definitions can be entered, windows in which the extant DoNaLD and Scout definitions can be systematically recorded and retrieved, a command history, and a screen in which the artefact under development is presented.

The history of the development of the simulation in Figure 1 illustrates some features of the process. Much of the simulation was developed by the first author in November 1992 over a period of 3 weeks or so, during which time the model was incrementally constructed whilst running as a background process on a workstation. At this stage, the model comprised some 2000 lines of code, including about 1000 definitions. Interactive

development sessions typically involved the addition of a group of definitions to attach a new visual component to the model, or of procedural actions in EDEN to simulate the introduction of new agents. It was occasionally necessary to reload the current files of definitions and actions (for instance, in the event of bugs in the interpreter, or system maintenance beyond our control), but several days typically elapsed between such events.

The development exercise in some ways resembled conventional engineering. It was sometimes useful to develop portions of the script independently, or to trace problems by extracting pieces of the script and exercising them in isolation. The layout of the statechart itself was developed by first describing the disposition and interconnection of boxes abstractly without supplying coordinate information, then adding coordinates derived by imposing a grid upon Harel's statechart. The process of tracing anomalous behaviour in the simulation was assisted by the close correspondence between the variables in the model and meaningful observations of the watch. For instance, at one stage, the watch was inappropriately programmed to chime whenever the minute fields of the display registered "00". Judiciously chosen experiments helped to determine whether the simulation was faithful to its intended behaviour. Ironically, the faithfulness of the simulation once led to the model having to be reloaded, when – in the course of such an experiment – the watch was inadvertently set to tick forward for 3 hours (a procedure that at that time required about 9 hours for its execution, and could not be interrupted).

Details of the statechart, and the button interface were added by the second author in December 1994, and Figure 1.2 added subsequently. Control over dependency and agency make it possible to take account of new views of the design process as they arise. For instance, it is easy to conceive how a traditional spreadsheet might be added to the model to relate the power consumption of the LCD display to the pattern of use. In a similar fashion, our model has recently been enhanced to include the chess clock in Figure 2, so that the models in Figures 1 and 2 can now be run together in real-time. In the chess clock it is appropriate to bind the motions of the second and minute hands together indivisibly (cf Figure 2). The simplicity with which we can adapt alternative computer models on-the-fly contrasts with the considerable ingenuity that would be required to modify a clock mechanism. And as the digital watch illustrates, the computer offers a potential for devising artefacts that could scarcely be imitated at all by a traditional mechanism.

Conclusion

The use of computers to construct cognitive artefacts has fundamental importance in modern applications. Empirical Modelling offers a powerful framework of concepts and tools to support the construction process. Further research and development is currently in progress to extend our principles and tools to take more explicit account of issues that arise in the concurrent engineering process [1], such as the representation of generic structures, constraints, commitments and agent privileges.

Acknowledgements

We are indebted to Simon Yung for developing the tkeden environment, and to Dominic Gehring for his helpful comments. We are also grateful to the EPSRC and Matra Datavision for sponsorship via grant GR/J13458 and a CASE studenship award.

References

1. V D Adzhiev, W M Beynon, A J Cartwright, Y P Yung *An Agent-oriented Framework for Concurrent Engineering* IEE Digest 1994/177, 9/1-9/4
2. D Harel *Algorithmics* Addison-Wesley 1987
3. D Harel, *Biting the Silver Bullet: towards a brighter future for Software Development* IEEE Computer, 1992
4. D Harel On Visual Formalisms CACM 1988, 31(5), p514-530
5. T Winograd *From Programming Environments to Environments for Designing* CACM, June 1995, 38(6) p65-74
6. Rumbaugh et al *Object-Oriented Modelling and Design* Prentice-Hall Int 1991

Figure 1.1: The Digital Watch

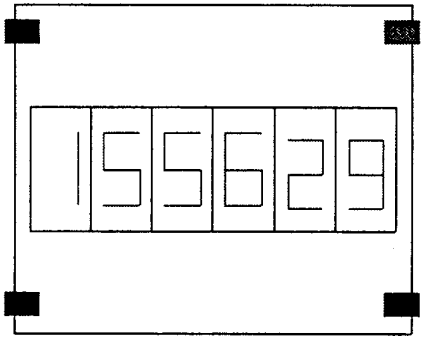
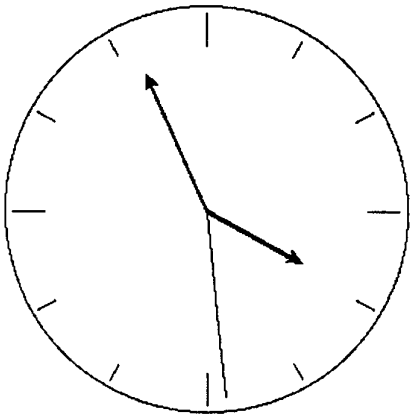


Figure 1.2: The Analogue Clock



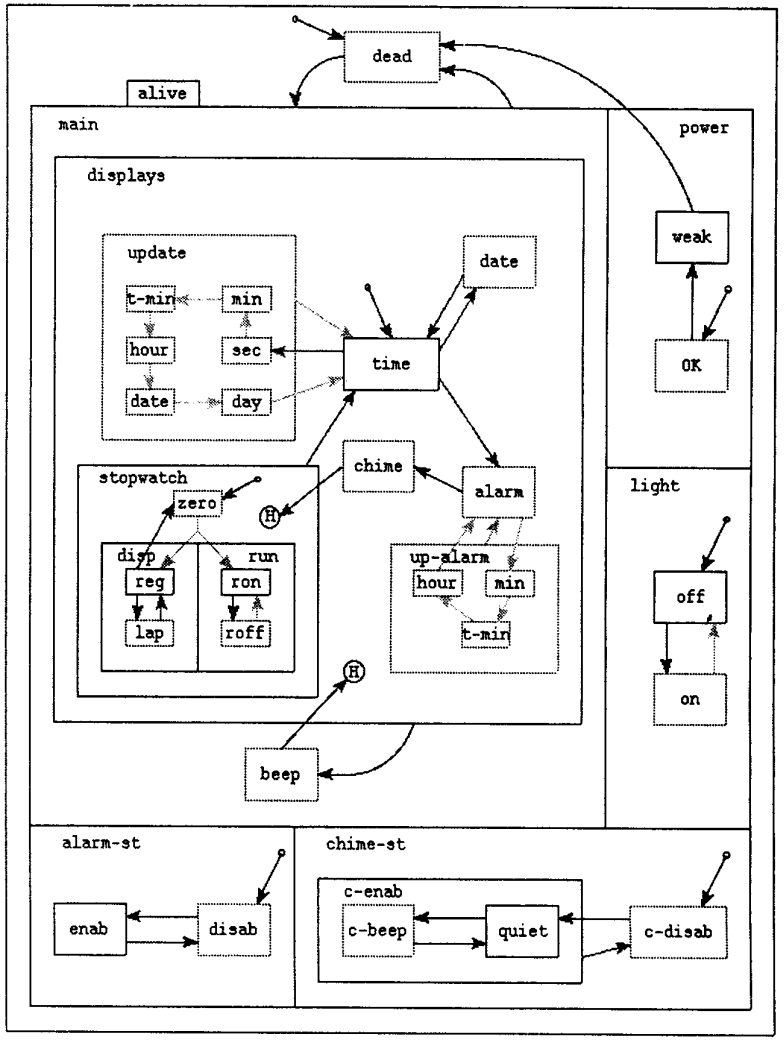


Figure 1.3: Harel's Statechart for the Digital Watch

Figure 1

Listing 1: An outline LSD specification for the Digital Watch

```
agent power() {  derive power_s = battery_charge(time)     // three-valued 3,2,1  }

agent watch() {
    derive      LIVE = (power_s >= 1)
    oracle      power_s
    agent main() { ... }              // as specified below
    agent alarm_st() {
        derive      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
    }                             // !d here means that button d has been pressed
    agent chime_st() { ... }
    agent clock() { ... }
    agent stopwatch() { ... }     // and several other agents, such as light() etc
}

agent main() {
        derive      LIVE=LIVE_watch
        oracle      LIVE_watch, main_s, alarm_s
        state       main_s = D        // an integer, viz. 1: Displays, 2: Beep||displays
        handle      main_s
        protocol    (main_s==D) & (time==set_time) & alarm_s==E -> main_s=B
        agent displays() {
            derive      LIVE=LIVE_main
            oracle      LIVE_main
            state       displays_s = T    // 1: Time, 2: Update, 3: Date, etc
            handle      displays_s, update_s, upalarm_s
            protocol    displays_s==T & !c -> update_s=1,
                        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
            agent disp_date() {
                derive      LIVE = LIVE_displays & displays_s==D, "watch_display = date as of clock()"
                oracle      LIVE_displays, displays_s
            }
            agent disp_time() { ..... }
            agent disp_upalarm() {
                derive      LIVE = LIVE_displays & upalarm_s>0,
                            displays_s = (upalarm_s==0%4)?A:UA
                            "watch_display = time as of alarm setting with appropriate 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=A,
                            !c -> upalarm_s++,
                            "event -> update set_time so as to increment highlighted digit in set_time"
            }
            .....................
        }
        agent beep() {
            derive      LIVE=LIVE_main & main_s==B
            state       main_s
            protocol    beep_stop -> main_s = D
        }
}   // end main()
```