Empirical Modelling Principles in Application Development for the Disabled

W M Beynon, R I Cartwright

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

Our ability to develop systems to meet special needs is influenced by several factors. The importance of emerging low cost computer technology cannot be denied, but this is not itself enough. In this paper, we are mainly concerned with the complementary role played by principles for computer programming and software development in exploiting technologies for the disabled. An abstract justification for our emphasis can be found in analyses of software engineering by Brooks [1] and Harel [2], which suggest an intimate connection between the challenges faced by modern software development and the broader problems of specifying and simulating complex systems of interacting human, electronic and mechanical components ("reactive systems"). From a more pragmatic point of view, the trend towards ever higher hardware specifications at ever lower cost has yet to be matched by comparable growth in the power and diversity of computer applications. Despite cheap computer technology, it remains difficult and costly to develop radically new software packages, or even to adapt existing software in comparatively modest ways. In practice, a new software product may be viable only where there is a mass market, because of the large investment in software development involved. Such issues are particularly relevant to the development of technology for the disabled, in view of the unusually high degree of innovation and customisation this typically demands.

An ongoing programme of research into computer-based modelling at the University of Warwick has led to the development of new principles and software tools that we believe can address some of the problems associated with more rapid and flexible exploitation of emerging computer technology. Many case studies in a wide variety of application areas have been investigated, both through sponsored research and through student projects at undergraduate and postgraduate level. The aim of this paper will be to highlight those aspects of our modelling method ("Empirical Modelling") that seem most relevant to developing software and hardware technology to meet special needs.

1. LSD Specification for Reactive Systems

Developing environments for the disabled typically involves combining several technologies, and explicitly taking account of the processing capabilities of the user. Systems in which there are many components of diverse types interacting concurrently are known as reactive systems. In this context, it is the complexity of the interaction, rather than the volume of data being processed, that is problematic. Our development method is oriented towards reactive systems, and explicitly involves the analysis and metaphorical representation of the processing capabilities of all state-changing agents in the system, whether human, electronic or mechanical (cf. [3]). In this analysis, the LSD notation is used.

In developing an LSD specification, the behaviour of the system is construed in terms of agents and the features of the system state to which these agents react and through which they operate. Adapting the terminology used to describe a scientific phenomenon, the salient features of a system are referred to as "observables". In our context, observables not only include features that can be perceived directly by human agents, but quantities that can only be indirectly measured, and characteristics of the system state that can be directly apprehended only through more sophisticated cognitive processes (cf. [5]). The novelty of our approach relies upon representing concurrent system behaviour in terms of what from an empirical standpoint can be (if perhaps only figuratively) directly perceived by agents. For instance, a car driver may not only see that the traffic light is red, but also - at the same instant (or at any rate "at the speed of thought") - recognise that it is appropriate to bring the car to a halt. Likewise, if the brakes of the car are in working order then pressure on the footbrake is immediately communicated as a braking force on the wheels.

By associating such observables with agents within the system, such as the car driver, the brake, and the wheels, and classifying them according to how they influence the operation of each agent, an LSD specification allows us to model different scenarios of interaction at many levels of abstraction. For instance, if 'seeing a red light' and 'recognising a halt signal' are regarded as potentially distinct observables, it becomes possible to distinguish between scenarios in which a car driver is visually handicapped (e.g. by colour blindness, or by an obstruction) or mentally confused (e.g. ignorant of traffic conventions, or drunk). In the same spirit, if the model for braking distinguishes between 'pressure on the footbrake' and 'braking force on the wheels', it is possible to investigate the different implications of faulty brakes, and ice on the road.

The potential benefits of agent-oriented analysis using LSD in the design of systems for the disabled lie in the flexibility it allows for considering alternative scenarios and the insight it can give into the precise assumptions that are being made about the environments in which agents operate. Much commercial software development is driven by the optimisations that can be achieved through focusing on closely circumscribed user-hardware configurations, for which reliability can be guaranteed, and where interaction follows carefully preconceived patterns. In that context, the subtle distinctions between one mode of interaction between agents and another

are typically irrelevant, but – on the other hand – there is little flexibility for changing requirements, or provision for exceptional behaviour. In programming for the disabled, particular care may need to be taken about exceptional conditions, for instance, in order to be sure that a system remains fail-safe.

2. Animation from an LSD Specification

The potential advantages of LSD specification can only be exploited subject to more subtle methods of developing an operational semantics than conventional methods presume. It is certainly not possible to construct an operational model from an LSD specification in the semi-automatic formal manner that it is possible to convert a high-level program specification into code. The way in which agents and subsystems are intended to operate is represented within our approach by constructing computer-based artefacts with which the system designers can experiment to evaluate and enhance their current design conception.

LSD specifications are first animated through constructing environments to represent interaction within fragments of the system. There is a close analogy with the experimentation and protoyping that an engineer performs in the design process. In Empirical Modelling, this preliminary design activity is captured using computer models based on imitating the values of the key observables within a system, typically by using visualisation. For instance, geometric analogues can be used to depict how far the footbrake is depressed, and what forces act on the car wheels, whilst the status of the traffic lights and the driver's interpretation can be displayed in diagrams with textual annotations.

The artefacts developed for experiment and exploration are constructed in a modelling environment that has been specially developed for Empirical Modelling. This environment is based upon an interpreter (tkeden) that allows dependencies to be established between the geometric elements in line drawings, the attributes of windows, and various basic data types, such as scalars, strings and lists. The state of a model developed in this environment is defined by a family of definitions that serve to interrelate the values of internal values and their visualisations. This mode of state representation resembles the network of dependencies between spreadsheet cells. In this environment, the operation of agents is animated either through the direct intervention of the modeller, in the guise of an omnipotent super-agent, or through triggered actions in the form of procedures that modify definitions in response to changes to values of particular variables.

Appropriately used, our modelling environment is well-matched to the basic requirements for experimentation with agents and interaction protocols. The propagation of dependency through a network of definitions is an excellent means to represent the way in which a single change of state (such a change in the colour of a traffic light) can be observed and interpreted in many ways at one and the same time. Concurrent action of agents can be appropriately represented by simultaneous modification of several definitions. Such simultaneous interaction may lead to conflicts that cannot be resolved without recourse to agency or dependency at a higher-level of abstraction.

There are a variety of ways in which it is possible to deal with interaction between agents at a more abstract level. Agent models can be animated from annotated LSD specifications through using an appropriate abstract machine model as a front-end. The tkeden environment includes an interface through which the definitions and values of variables can be readily retrieved and monitored. Further research is currently addressing more sophisticated organisation of scripts of definitions through forms of higher-order definition. In combination, these developments can potentially provide a powerful framework for co-operative product design [4].

3. Practical Advantages and Prospects

Model construction in the Empirical Modelling framework is a human-centred activity in which the emphasis is upon human-computer co-operation, and the central theme is the exploration and interactive adaptation of a computer-based artefact that reflects evolving insight into the system requirement [7]. Our methods favour open development rather than closed-world modelling, so that at any point during the modelling process a user can either enhance or modify a model (e.g. to include more detail), change their viewpoint of the model (e.g. add new parameters to their interface) or change their mode of interaction (e.g. substitute a slider bar for a textual representation of a number). Our experience has shown that, in many contexts, reasonable performance can be obtained from our software prototypes without compromising such flexibility for change. Where special-purpose hardware devices are involved, it is in general necessary to use our methods to construct a simulation software prototype. Where appropriate, it is then possible in principle to generate conventional software and hardware specifications automatically from such models.

Our modelling method potentially offers good prospects for enhancement and customisation of models by non-specialist users. The modelling process relies on incremental enhancement of a computer model, based on observation of real-world objects that can be simple (often trivial). This is illustrated in making a computer model of an analogue clock [6]. Turning the control on the back of a ticking analogue clock results in the time displayed being altered and discloses a direct relationship between the position of the minute hand and the hour

hand. When this relationship is expressed in a script of definitions, the user of the computer model can directly and realistically imitate the process of resetting the time. It is by a sequence of such simple steps that the model is incrementally developed as a modeller's insight into the referent evolves. The nature of this modelling process favours a shift of emphasis, whereby disabled users can be more independent of programmers, and be in a position to gain understanding of how a system has been constructed.

In developing software and designing tools to assist the disabled, it is desirable to involve the potential users in the creative process at every stage – from conceptual design through to prototype, revision and eventual mass production. This demands the inclusion of disabled designers in design and testing teams. Such designers need to be able to set up their own view of the programs or artefacts being considered so that they can monitor the interactions of others in the team and present their own view of the model in discussions. This may be particularly important where the disability of a designer prevents them from accessing a computer-based model built using a conventional software package. Our tools admit several different viewpoints on the same model, allowing all designers to tune interfaces to suit their own specific interests.

Interaction with our models can go beyond the simple stimulus-response patterns characteristic of much conventional user-computer dialogue. It is easy to enhance a model by introducing monitors to alert the user to potentially undesirable conditions, and to give users discretion in their response to monitors. For instance, when a monitor indicates that a value is outside a specified range, the user can either respond by changing values in the current state to satisfy the boundary conditions, or by redefining the range of acceptable values.

The input and output from a model can be configured to use many different types of multi-media device. Experience of parameters in a model could be through connections to audio devices or electic motors producing a vibration rather than the more commonly used VDUs. Inputs to change the state of the model could be configured to work using touchpads, dial boxes or voice activation rather than a keyboard and mouse. All the devices can be configured to connect to the model without making any change to the model itself and this customisation can be saved and re-used to connect to other models in the future. This potential has been noted by Nardi in her study of end-user programming with multi-user spreadsheets [8].

User-interfaces help to establish consistency between applications. For instance, a common convention in many applications is to locate the **Exit** or **Close** method for a program in the **File** pull down menu. Uniform modes of access are typically important for a disabled person, for whom reusing existing skills is far more efficient than developing new ones. Uniformity in interface design can be represented in the construction of reusable device scripts specific to a user. Because of the incremental nature of our development process, a teacher or amanuensis can configure a model or application to the point where particular disabled users can themselves customise the application more closely to their own needs.

In a conventional software development, it is necessary to recompile a Graphical User Interface before the view of a model is changed, and this typically means reinitialising the model. In contrast, Empirical Modelling makes no essential distinction between changing the state of the interface and changing the state of the model. Either can be modified on-the-fly without interference. There is no need to circumscribe any part of the model before or during its construction, and if circumscription is required it can be provided in the form of monitors rather than embedded in choices of data structures with preconceived restrictions and exceptions.

References

1. F P Brooks, No Silver Bullet: Essence and Accidents of Software Engineering, Computer 20:4 (1987), 10-19
2. D Harel, Biting the Silver Bullet: Towards a Brighter Future for System Development, IEEE Computer, January 1992, 8-20
3. W M Beynon, I Bridge, Y P Yung, Agent-oriented Modelling for a Vehicle Cruise Controller, Proc. ESDA Conf., ASME PD-Vol. 47-4, 1992, 159-165
4. V D Adzhiev, W M Beynon, A J Cartwright, Y P Yung, An Agent-oriented Framework for Concurrent Engineering, Proc. IEE Colloquium: Issues of Cooperative Working in Concurrent Engineering, October 1994, Digest 1994/177, 9/1-9/4
5. W M Beynon, M S Joy, Computer Programming for Noughts and Crosses: New Frontiers, Proc. PPIG'94, Open University, January 1994, 27-37
6. W M Beynon, R I Cartwright, Empirical Modelling Principles for Cognitive Artefacts, Proc. IEE Colloquium: Design Systems with Users in Mind: the Role of Cognitive Artefacts, Digest 95/231, 8/1-8/8
7. W M Beynon, S B Russ, Empirical Modelling for Requirements, CSRR#277, Computer Science Dept, University of Warwick, Sept. 1994
8. B Nardi, A Small Matter of Programming: Perspectives on End User Computing, The MIT Press, Cambridge, MA 1993