A conception of computing technology better suited to distributed participatory design

Meurig Beynon　　Zhan En Chan
wmb@dcs.warwick.ac.uk　　*echan@dcs.warwick.ac.uk*
Department of Computer Science, University of Warwick
Conventry CV4 7AL, United Kingdom

Abstract

Distributed participatory design aspires to standards of inclusivity and humanity in introducing technology to the workplace that are hard to attain. The demands it makes upon the development and use of computing technology are particularly topical, as the potential for automation and distribution through embedded and mobile devices continues to develop. Standard views of computation propose ways in which to interpret all products of computing as *programs*, but give limited conceptual support for understanding computer-based design artefacts whose role in communication and elaboration eludes capture in a functional specification. This motivates our brief account of the alternative conceptual framework for computing afforded by *Empirical Modelling*, a body of principles and tools that can be applied to the development of a variety of computer-based artefacts relating to analysis, design and use that are most appropriately interpreted as *construals* rather than programs. The paper concludes by hinting at some of the ways in which developing construals using Empirical Modelling can assist distributed participatory design.

1 Introduction

Distributed participatory design (DPD) is concerned with design processes in which the stakeholders have different levels of expertise and competence and are located in different environments. The priority for this activity, as identified by Crabtree [7], is the creative use of technology to improve working practices in such a way that it does not destroy the workers' skills, does not take away their autonomy, and enhances their quality of life. Where the introduction of technology is concerned, DPD poses specific major technical challenges:

- Integrating the human and the technological: DPD demands an intimate integration of the technological infrastructure with human activities. As illustrated in Crabtree's account of the UTOPIA project [5], the level of automation may not in fact be high; finding the most effective role for automation does not necessarily involve making radical changes to existing workpractices.
- Enabling flexibility and evolution: DPD requires development techniques that make it possible to shape technology in a flexible and open-ended manner. Since participatory design engages with the entire development lifecycle (cf. [10]), it should ideally be possible to take the perspectives of analysts, developers and users into account. This favours a conception of technology that is evolutionary in nature and accomodates elements of customisation and on-the-fly modification.
- Supporting diverse interaction and communication: DPD highlights the role of technology in support of human interaction and communication rather than simply as a means to achieve goals in ways that are more efficient and cost-effective. The type of interaction and communication to be supported has also to be highly diverse. For instance, in addition to providing support for the workpractices to be implemented, it should also enable the role-shifting and exploratory investigation involved in the development, where all stakeholders have an interest.

The primary focus of this paper is on the impact that two different underlying conceptions of computing technology have on distributed participatory design. In the sections which follow, our objective is

- to highlight ways in which a traditional conception of computing technology obstructs the integration of human and computer-based activities in support of DPD;
- to outline an alternative conception of computing that offers principles and tools much better aligned to the emerging practice and aspirations in DPD.

2 Duality in the conception of computing technology

The traditional way in which computing applications are conceptualized has deep historical roots. In using a computer as a calculator or data processor it is entirely appropriate to think in terms of categories such as: articulating the requirement and identifying it as a computable function; specifying this function in abstract high-level terms that can be understood by the programmer; translating this into code that can be interpreted by the machine; devising a protocol by which the user can supply parameters to the function and be presented with an output. The range of applications for computing has of course changed radically since such input-output processing was the dominating practical concern, but there has been no comparable conceptual shift in "understanding the fundamental nature of computing". This is not an issue that is of sole concern to the computer scientist; the pervasive role of computing, and the limited nature of our conceptual grasp, has arguably had a broad impact on our understanding of systems that combine human and automated agency.

The traditional conception of *computer program* is particularly ill-suited to the technological challenges in DPD, as identified in the introduction:

- Integrating the human and the technological? In the classical view, an executing program is a means to an end in carrying out a computation – human activity and programs intersect at preconceived rendezvous points for stereotypical input-output interaction. At these points, the human has to act in highly constrained preconceived role as a 'user'. Activities such as "interrupting the program execution", or "modifying an executing program" are outside the conceptual scope.

- Enabling flexibility and evolution? In the classical view, the program requirement, its specification, its code and its user interface are different species, each with its own specialist human interpreter – the analyst, the designer, the programmer and the user. The fundamental idea behind the classical program is that once its function has been clarified and specified, every possible ingenuity and optimisation can be exercised in carrying out this function, thus saving computer system resources and user time. If you change the requirement, you may well need to ditch the existing program, since ingenuity and optimisation is typically highly function-specific.

- Supporting diverse interaction and communication? The classical program is intended to support a black-box style interaction. It is not considered important for the user to understand what the program is doing internally, nor for the programmer to know what the user is doing beyond awaiting the computation of a specific function as mediated through a specific interface. Within the classical framework, it is possible to imitate cross-over interaction between different species of program specialist – for instance, to allow the user to adapt their interface, or customise the program execution for another user. But in so far as a computing application is viewed as a program, it has a fixed function that determines the way in which relationship between its execution and its surrounding context is to be interpreted, and this limits its expressed potential for communication.

The fact that the classical program, as represented above, is no more than a parody for what modern software technology discloses is not the issue. Of course, advanced user interfaces and interface devices, databases, spreadsheets, agent technologies and the like have raised entirely new logistic, experiential and real-time issues that have transformed practice. Ways of thinking about computing technology are quite as influential as – if not more influential than – the technologies themselves. The influence of goal-orientation and programmed interaction remains prominent in research fields such as CSCW, user-centred design and agile development. It promotes a pernicious duality that separates the users of an application from its developers and partitions the life-cycle into distinct kinds of activity that are hard to integrate. This is in conflict with one of the central notion of DPD: that of treating workers as total human beings rather than merely *users*.

In dissolving the duality at the heart of classical thinking about computing, it is helpful to consider the distinction between two kinds of activity:

- identifying the potential users and functionalities for an application;

- gaining familiarity and understanding in the domain in which an application is to operate.

The concept of gaining familiarity with a domain is broader and more general than the identification of applications and users – it may not even presume any pre-existing notion of "user" or "application". It is also clear that the context for performing a task can be configured in such a way that its execution requires very little understanding of the domain. Indeed, as discussed at length in [9], the importance of context in relation to task is a profound and long-established strand in Chinese thought that finds its expression in the concept of *shi* as an"inherent potentiality at work in configuration" ([9], p.14). The computer programming context can be regarded as an archetypal context that is contrived so that effective and meaningful action can be mediated by what Bornat describes as "completely meaningless" programs [6].

Two quite different viewpoints on design activity are highlighted here. One puts its primary emphasis on clarifying the tasks to be carried out and bringing maximal ingenuity to bear on their efficient implementation through automation. The other focuses instead upon a less goal-oriented exploration of the potentialities within the domain of application. The first approach is associated with the traditional conceptual framework that leads from requirements analysis of the workplace to specification, identification of function, implementation and maintenance. This framework involves an explicit framing of the notions of users and roles. The second approach involves a more open-ended holistic investigation of the application domain that cannot be decomposed into conceptually distinct formal phases. In this approach, the configuration of the application context, the nature of the problems to be addressed, and the roles for users and automation emerge in parallel.

3 Empirical Modelling

Empirical Modelling (EM) is a body of principles and tools that has been developed to support a conceptually different way of thinking about computing activity that is oriented towards a more holistic approach to design. Its primary emphasis is on studying dispositions within a situation and creating configurations to exploit these, rather than identifying and implementing functions to achieve pre-specified goals (cf. [3]). To this end, it involves the development of computer-based artefacts that are more appropriately interpreted as *construals* than programs. That is to say, they are constructions made by the modeller that embody characteristic features of a situation that – like dispositions – are revealed through interaction.

In the DPD context, EM can be viewed as developing interactive situation models (ISMs) that help to capture and convey personal understanding of situations or phenomena. The referent for an ISM can be a real-world situation or phenomenon. It could also be an imaginary creation in the modeller's mind. The relationship between an ISM and its referent is mediated by the pattern of observables, dependencies and agencies that it embodies. The counterparts of *observables* and *dependencies* are *variables* and *definitions* within the ISM. The current state of an ISM is represented by the current set of extant variables and definitions – a *definitive script*. This script may change and evolve dynamically subject to no constraint other than that the state of the ISM continues to stand in a meaningful relationship to its referent as far as the modeller is concerned. The counterpart of an atomic agent action, whether this is carried out by the modeller or any other agent, is a change to the ISM that involves adding, deleting or revising a definition in the script. A change to the definition of a single variable is effected in such a way that all contingent changes to the values of other variables are carried out atomically, so as to achieve an "instantaneous" update.
The significance of an ISM cannot typically be appreciated in isolation from our contextualized interaction with it; in this respect, it is unlike a computer program, whose semantics can be captured in the abstract functional relationships that it serves to establish. The interactions that shape an ISM have much in common with the kinds of experimental interaction associated with explanatory artifacts that a scientist might make in the early ("pre-theory") stages of an investigation, or that an engineer carries out when speculating about some aspect of a design. A characteristic feature of EM is that what we understand by a *model* is identified with patterns of interaction with such an artefact that emerge from experimental and exploratory activity over a period of time. This activity is broad in scope: it may involve complementary interaction with the external situations and phenomena to which the ISM is intended to refer. It is also open-ended and potentially subjective in nature: the modeller's conception of an ISM and its associated referent is subject to evolve as the modelling activity progresses. In comparison with traditional mathematical models, an ISM is intrinsically fuzzy, soft and fluid. If its relationship to its environment can be sufficiently well-engineered, however, it may – at the discretion of the modeller – be exercised in a way that it emulates a model that is precise, hard and tightly specified (cf. [4]).

The principal tool for Empirical Modelling is `tkeden`. The tool supports a variety of notations in which definitions to express dependencies amongst variables of various different types can be formulated. (For instance, variables may correspond to scalar quantities and attributes, strings, geometric elements like points and lines, components of a screen display, relational tables etc.) A distributed variant of `tkeden`, called `dtkeden`, can be used to support modelling in a distributed environment. This provides an open environment for developers and users to negotiate a shared interpretation and shared representation. Negotiation of meaning and consensus from many different personal, social and cultural perspectives is at the core of collaborative working [1]. The meaning that is attached to an ISM is determined not by a formal computational semantics, but through the interactive experience it offers to each human participant [2]. The distributed variant of `tkeden` offers the developer instant feedback, since any redefinition has an immediate impact on the state that is either directly visible to the developer (e.g. changes to the attributes of display components) or directly affects the disposition of the script to respond to future interaction (e.g. introducing a dependency to link the motion of a lever to the configuration of an object). As with traditional prototypes, this enables "potential users" to exploit an ISM in sense-making activities. The modelling of behaviours as realised through first modelling dispositions (though typically consuming more attention and time and requiring more computational resources) is a much more expressive activity than directly prototyping behaviours. It is also an activity that allows users and developers to interact through the modelling environment, whether through tweaking the ISM or redefining components entirely. This affords real-time collaboration among developers and users beyond document-centric communication.

By way of an indicative illustrative example, the Virtual Electronics Laboratory (VEL) was developed collaboratively by two MSc project students at Warwick [8, 11]. The model was directed at teaching elementary electronics both in the classroom and in a distributed environment. In developing the VEL, **D**'Ornellas and **S**heth first created an environment in the form of a definitive script to support a wide variety of interactions and behaviours. D focused on enabling dependencies between complex matrices that defined the mathematical semantics of circuits; S on setting up dependencies amongst interface components. Collaborative interaction between D and S was enabled by identifying a small set of key observables and exploiting dependency to bridge two contrasting modes of observation, one concerned with primitive elements to be assembled into an electronic circuit, the other with graphical icons to be manipulated on the screen.

In the early stages, scripts for the internal semantics and the interface were developed independently on stand-alone `tkeden` interpreters. As the conceptual states of the construal became ever more rich, so the definitive script became more complex and integrated. For instance, where interface and semantics were initially viewed in isolation, they were afterwards combined, first in bringing conceptual integrity to modelling electronic circuits, then later in representing the states associated with different scenarios of use. In this process, the emphasis shifted from the configuration of the script itself to the varieties of meaningful agency that could be effected through redefinition. In practical terms, the unbounded diversity in potential agency was reflected in diverse ways of manipulating the state of the model: by directly changing the script via the `tkeden` input window; through interface actions that triggered specific redefinitions; and through different modes of redefinition and interpretation built into the distributed `tkeden` interpreter itself.

Unusually rich semantic possibilities are afforded by such a representation of conceptual state. From the conventional perspective on possible 'uses' of the model, the teacher could set up a circuit, and broadcast this to the class. Students could work individually to adapt a copy of a circuit supplied by the teacher. A class activity could be set up whereby the teacher dynamically authorised groups of students to change different circuit components, and the impact was displayed on a public model. This is merely to hint at what redefinition of the script can achieve, which could encompass modifying the mode of output from a phase graph to a numeric representation, or integrating the model with a traffic lights simulation. There are also potential uses for the way in which a script can be viewed as capturing the history of model interaction, construction or revision.

In summary, the VEL development exhibits many features relevant to DPD. In keeping with the holistic nature of EM, it is hard to identify how each of the specific key issues introduced above is represented in isolation – integration of the human and the technological, support for evolution and diverse modes of interaction and communication are bound up together. Through this paper, we hope to draw the attention of the DPD community to the obstacles that the classical view of computing puts in the way of such a holistic integration of concerns, and arouse interest in the new possibilities that EM affords.

References

[1] M. Beynon, S. Russ, and W. McCarty. Human computing: Modelling with meaning. *Literary and Linguistic Computing Journal*, 21(2):141–157, 2006. Oxford University Press.

[2] W. M. Beynon. Radical Empiricism, Empirical Modelling and the nature of knowing. *Pragmatics and Cognition*, 13(3):615–646, 2005.

[3] W. M. Beynon, R. C. Boyatt, and S. B. Russ. Rethinking programming. In *ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)*, pages 149–154, Washington, DC, USA, 2006. IEEE Computer Society.

[4] W. M. Beynon, J. Rungrattanaubol, and J. Sinclair. Formal Specification from an Observation-Oriented Perspective. *Journal of Universal Computer Science*, 6(4):407–421, 2000.

[5] S. Bødker, P. Ehn, J. Kammersgaard, M. Kyng, and Y. Sundblad. A UTOPIAN experience: On design of powerful computer-based tools for skilled graphic workers. In G. Bjerknes, P. Ehn, and M. Kyng, editors, *Computers and Democracy–a Scandinavian Challenge*, pages 251–278. Aldershot, Gower, Avebury, England, 1987.

[6] R. Bornat. Is 'Computer Science' science? In *European Conference on Computing and Philosophy (ECAP)*, Norwegian University for Science and Technology, Trondheim, Norway, 2006.

[7] A. Crabtree. *Designing Collaborative Systems: a pratical guide to ethnography.* Springer-Verlag, London, 2003.

[8] H. P. D'Ornellas. Agent oriented modelling for collaborative group learning. MSc Project Report, Department of Computer Science, University of Warwick, Coventry, United Kingdom, September 1998.

[9] F. Jullien (Translated by Janet Lloyd). *The Propensity of Things: Toward a History of Efficacy in China.* Zone Books, New York, 1995.

[10] B. A. Farshchian and M. Divitini. Using email and www in a distributed participatory design project. *SIGGROUP Bull.*, 20(1):10–15, 1999.

[11] C. R. Sheth. An Investigation into the Application of the Distributed Definitive Programming Paradigm in a Teaching Environment: The Development of a Virtual Electrical Laboratory. MSc Project Report, Department of Computer Science, University of Warwick, Coventry, United Kingdom, September 1998.