The Computer as an Agent in Empirical Modelling

0415747
Weighting: 50% paper, 50% modelling

Abstract

I modify an existing educational computer simulator to behave as an agent in an Empirical Modelling context, and present an example, in the form of a simple Empirical Modelling study, of how such an agent can form part of a practical model. I then suggest ways in which the aforementioned concepts can be applied within the contexts of engineering and education.

1 Introduction

It is hard to buy an electronic device nowadays that does not come with an embedded microprocessor. These applications typically use a microprocessor to control physical devices with or without moving parts – starting or stopping a motor in a washing machine or illuminating light emitting diodes on a control panel. They also react to the actions of a user via a control panel or to the state of sensors[1].

These applications usually require a procedural program to be written that write appropriate data or control commands to a specific port or location in memory. In order to program the hardware, a programmer must either have the device ready built, or must have a dedicated emulator written that can be used while the hardware is unavailable. The same is true of teaching such skills. The first year Computer Organisation and Architecture module of the University of Warwick's Computer Science MEng and BSc courses make use of a piece of dedicated hardware called the *SWET*[2]. This provides an embedded computer environment that can be interfaced with other real-world hardware.

While such educational tools are useful within a laboratory setting, I argue that it is not so useful for self-study at home or in other, less well-equipped laboratories; or for schools teaching Electronics or Computing who might not be able to afford to develop such tools.

The aim of this paper is to discuss how well Empirical Modelling can support the production of hardware emulation tools – and particularly its suitability for emulating the kind of hardware that might be used in a laboratory setting on a user's computer screen.

[1] for instance, the author recently used a washing machine that automatically adjusts its cycle according to the mass of garments placed in the drum

[2] *Super Warwick Education Tool*

This is illustrated with a C extension that integrates Eden with the *SCICS* tool – a program written in C by Yung (1995) which stores and visualises machine state in the Unix *Spreadsheet Calculator* (`sc`).

Using an Eden model with dependencies on the SCICS machine's state, I illustrate how a fictitious computer running as a virtual machine can perform the role of an agent within an Emprical Modelling model.

2 Background

Beynon (2007a, p.97) describes the "methodology for modelling with dependency" in terms of a context, in which there are three key components:–

The Construal The representation of the model by the computer. This is the way that interaction is facilitated by the model – the "observables that we deem to mediate [agents'] interaction."

The Referent The artefact that the modeller is modelling.

Understanding The modeller's understanding of both the Construal and the Referent.

In this paper, the distinction between the roles of the Referent and the Construal are blurred, considering the case where a simulated computer, fictitious or otherwise, can be considered to be part of the referent, rather than purely maintaining the model's construal.

The idea of having a formally-defined artefact as a referent is not new, and empirical modelling has been used previously to simulate such things. For example, Care (2005) used Eden, Scout, Donald and Sasami to simulate and model the operation of an analogue computer known as the *planimeter*. While by the nature of the planimeter its operation is defined entirely by formal rules of physics and engineering,

I believe the use of Empirical Modelling allows the user to investigate its operation under modification or even breakage much more easily than would be possible with a traditional program.

The ability to build the model without knowing the full details of how the system works also makes Empirical Modelling an ideal tool for modelling this kind of artefact. When working with historical sources, this is likely to be the case.

Another referent that could easily be (and frequently is) modelled by an ordinary procedural computer program is the popular logic game Sudoku. [3]

However, King (2006) built an EM model of Sudoku with the premise:

> "The purpose of this exercise was not to develop a computer program capable of providing the solution to any logically solvable sudoku puzzle; rather, it was to explore ways in which the puzzle solving exercise could be usefully supported by the computer and how the skills of a human solver could be embodied in a computer-based artefact."
> (King, 2006)

This might not be quite so easy to do within a traditional procedural or functional programming environment.

The ability to interactively extend existing models is also characteristic of Empirical Modelling, as was illustrated when Harfield and King (2007) extended the Sudoku model with support for colour to indicate, in what I believe to be a novel and useful fashion, the rank of the numbers that would best fit each cell given the current puzzle state.

The following sections discuss how a computer can become part of or interact with an Empirical Modelling artefact, and the usefulness of such interaction.

3 Development and Modelling Study

In order to illustrate the concepts of this paper, I first got SCICS running on the DCS Linux network and briefly studied its operation. My findings are presented in Section 3.1.

I then undertook the ambitious task of integrating SCICS with Eden, as discussed in Section 3.2, and explore an Empirical Modelling application of this in Section 3.3.

[3] An example of such an implementation is GNOME Sudoku (Hinkle, 2006).

3.1 Working with SCICS

The fictitious *Introduction to Computer Science* machine that was originally documented in 1979 and run in what was then the *University of Warwick Computer Centre*, as a machine for teaching "elementary hardware" and "machine-level programming" concepts. (Beynon and Buxton, 1979)

> "[SCICS] combines the UNIX terminal-based Spreadsheet Calculator sc and a simulator for the Introduction to Computer Science machine into one environment which then has definitive (spreadsheet) and procedural (machine) facilities."
> (Yung, 1995, README.xml)

The difference between SCICS and traditional machine emulators is that the storage used is a spreadsheet, rather than a block of memory invisible to the user. By running and studying the Heapsort example provided by SCICS, it was found that the spreadsheet can be used to visualise certain properties of the machine state using spreadsheet formulae (and hence definitive programming). In the case of Heapsort, column `G` of the spreadsheet is used to display whether or not the heap condition holds in their respective data cells in column `F`.

The student using that example to model the heapsort algorithm could easily define more formulae in unused cells to visualise other properties of the system.

In the following section, the method by which the SCICS and Eden modelling tools have been integrated to provide a richer modelling environment than a spreadsheet can provide.

3.2 EDICS – Integrating SCICS with Eden

This paper is all about the usefulness of building models around a simulation of a computer that executes a fixed program and shares its state with an empirical modelling environment. To facilitate this, a the simulated computer can be considered to be an *agent* within a larger model. By allowing this agent to interact in some way with the state of the model, this section argues that the interation between other agents and a computer can be modelled.

To provide the enabling technology, SCICS was extended to allow it to update a subset of observables defined within an Eden model with the contents of its memory. The approach taken was to extend SCICS

with code that enables it to behave as an agent to a `dtkeden` server session. Due to time constraints, the ability for Eden to update the SCICS machine's memory was not implemented; however the current one-way communication is sufficient for the purpose of illustration.

The result of that work is that SCICS emulates a copy of `dtkeden` running in *client mode*. By running a `dtkeden` server in *Normal Mode*, and connecting using SCICS with the `-d` flag, SCICS sends commands to the server after each re-evaluation of the spreadsheet, defining observables named along the lines of `ics_A0` which corresponds to cell `A0` of the spreadsheet. 512 such observables are maintained, corresponding in numerical then alphabetical order to the 512 memory addresses an ICS page:

```
ics_A0   ics_B0  .  .  .   ics_P0
.          .                .
.          .                .
ics_A14  ics_B14 .  .  .   ics_P14
```

3.3 A Modelling Scenario using Eden and SCICS

So far, the SCICS tool has been tested and has been modified to share its state with a running Eden model. This section describes a simple modelling exercise that was performed to demonstrate the usefulness of having a computer operating as an agent within an EM modelling scenario.

Consider the *LED Digit* Donald model presented by Beynon (2007b, p.2), which simulates a standard 7-segment display, along with cell `F1` of the SCICS *heapsort* example. `F1` is the first data element in the 15-element array that is to be sorted.

Now consider the following scenario: A student has been asked to develop a device that will take a set of numbers as input and display the smallest of those numbers on an LED display.

The LED Digit model provides both the 7-seg display simulation and simulates the operation of a BCD to 7-seg decoder via the `digit` observable.

Assuming that the data is entered to the computer for us (by means of the SC spreadsheet in this case); by duplicating the 7-segment display twice, and making the `digitT` and `digitU` observables dependent upon Eden observables that are in turn dependent upon the observable `ics_F1`, it is possible to display the number (up to two figures) that is presented in that memory location.

The Eden observables `tens` and `units` correspond to their respective place values extracted from `ics_F1`, and provide the individual digits for the Donald openshape `leds` to display.

Figure 1 illustrates the configuration of observables, as well of the simpler definitions, are presented in the context of a block circuit schematic, representing the connections (dependencies) between a SCICS memory cell and the state of the visualisation (the 7-seg displays).

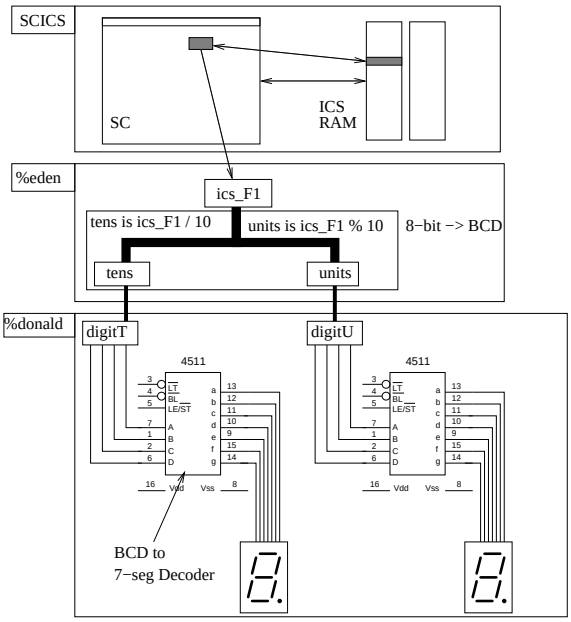

Figure 1: How to Drive a Simulated Digital Display from SCICS

Figure 2 shows SCICS having updated the Eden observable `ics_F1`. In Donald the value of `F1`, 42, is displayed in the simulated pair of 7-seg displays due to the dependencies defined in Eden. As can be seen from the screen shot, it is not necessary for the SCICS machine to be running a program in order for this to work. As a result, the modeller can change memory locations as part of the experimentation process without having to write a plethora of new ICS programs to update specific memory blocks.

4 Applications of the Work

The development and modelling exercise has shown, with a toy example, how a computer or even a simple spreadsheet can provide some kind of agency within an Empirical Modelling simulation.

The ability to model interactively using definitive scripts provides a rich environment in which students, instructors and engineers can experiment with sim-

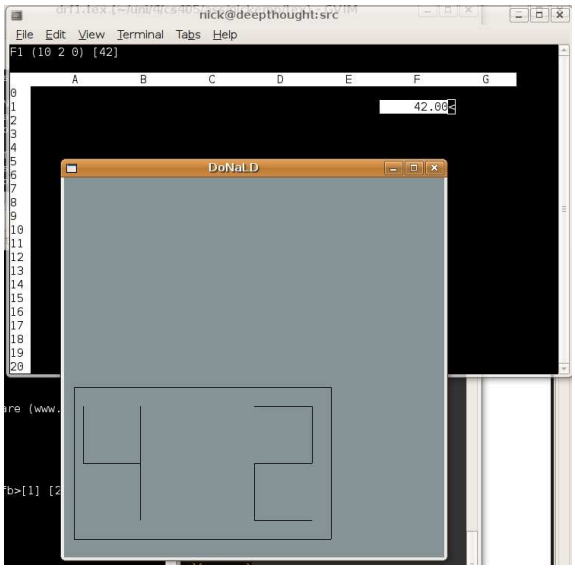

Figure 2: SCICS driving a Donald 2-digit display

ulated computer hardware, and incrementally build functional simulations and designs for existing or future computer peripherals.

Some of the applications of this are described in the following sections.

4.1 Empirical Modelling in Engineering

One such application is in the design phase of an engineering (specifically electronics) project that requires some sort of integration with a computer hardware, be it internal to a computer or external. By integrating Eden with a computer simulator (or even any other computer program or programming language) in the manner discussed, it would be possible to use Eden and its associated tools to create a visualisation of the hardware, and animate its operation under the control of the computer.

4.2 Teaching Students to Program and Design Computer Hardware

A notable advantage of modelling with dependency is the ability of the user to become part of the modelling process. In Empirical Modelling, building, modelling and extending models are experimental processes. The focus of the exercise is not only in using the finished model; but also in the production of the model itself.

While providing source code to a traditional program allows users to study and modify the code for their own needs, Empirical Modelling *encourages* and *supports* the user in doing so. The modelling process is interactive; and the approach of modelling by dependency rather than procedurally allows the user to see immediately the effect of making a change to the model. In traditional programming, the effects of making such changes are only visible the next time the program is run. This includes, albeit to a lesser extent, interactive programming languages such as Python where the user must still initiate some function and wait for its execution to complete in order to see the results of their experimentation.

The interactivity of both SCICS and Eden would allow an instructor to set students exercises both in the low-level programming of pre-prepared device simulations in Eden and in the creation of hardware simulations.

Using such tools would allow students to learn independently, outside of the laboratory setting.

5 Closing Remarks

In this report, I have shown that it is possible to combine concepts from both Empirical Modelling and traditional Computer Science, to produce an environment in which Computer Science students and engineers can incrementally build models around — and experiment with — concrete data structures and machines.

From a practical point of view, I have added code to the previously-existing SCICS program that allows it to manipulate the observables in a running dtkeden model; and produced a simple model illustrating how data provided by the SCICS spreadsheet can be used to visualise its state. I then went on to argue that the applications of this work can be much wider than simple state visualisation, and that similar work could be used for designing, simulating and writing software for real hardware projects.

5.1 Recommendations for Future Work and Application

5.1.1 Two-way Communication with Eden

The existing `eden_glue` code could be extended to allow SCICS to accept and parse commands sent to it by the dtkeden server. This would allow SCICS to query the state of the model to which it is an agent and update its state accordingly. Models could then be built that simulate the operation of sensors and human interface devices, and feed that information back into SCICS for processing by the running program.

This was not done along with this paper due to time constraints.

5.1.2 Integration with Other Software

While we used SCICS and the SC spreadsheet for illustration, there is no reason why a similar integration could be performed with other software, whether it be operating systems, desktop applications, a Unix shell or a programming library.

5.1.3 Bindings to Other Programming Languages

For illustration purposes I wrote a one-way C binding out of necessity, in order to integrate with SCICS. That does not mean that a Python, Ruby or Java binding could not be acheived or be useful.

5.1.4 Investigate Further Applications

Finally case studies could be performed in using this technique for solving real-world work.

Acknowledgements

I would like to thank Meurig Beynon for providing direction for my initial idea which has changed quite a bit since the initial provisional title was submitted.

Richard Myers has also been of great assistance, and the idea of using network socket programming (in contrast to my original intention to integrate Eden and SCICS directly using C) was inspired by my communications with him. This saved me a lot of headaches trying to decipher Eden's source code and virtual machine semantics.

References

Meurig Beynon. Computing technology for learning - in need of a radical new conception. *Journal of Educational Technology & Society*, 10(1):94–106, 2007a. URL `http://www.ifets.info/download_pdf.php?j_id=34&a_id=730`.

Meurig Beynon. Visualisation using Empirical Modelling principles and tools. Technical report, Computer Science, The University of Warwick, Coventry CV4 7AL, June 2007b. URL `http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/papers/downloads/103.pdf`. [Online].

W.M. Beynon and J.N. Buxton. The ICS3 Computer. The University of Warwick Computer Centre Report (CCR) 22, University of Warwick, July 1979. URL `http://empublic.dcs.warwick.ac.uk/projects/scicsYung1995/Docs/ICS3Computer/report.html`. Transcribed to HTML by Ashley Ward in May 2002.

Charles Care. Planimeter Models. Online, Feb. 2005. URL `http://empublic.dcs.warwick.ac.uk/projects/planimeterCare2005/`. planimeterCare2005.

Brian "Beej Jorgensen" Hall. *Beej's Guide to Network Programming. Using Internet Sockets.* Published Online by Author., Aug. 2007. URL `http://beej.us/guide/bgnet/`. Version 2.4.5.

Antony Harfield and Karl King. Sudoku Colour. Online, May 2007. URL `http://empublic.dcs.warwick.ac.uk/projects/sudokucolourHarfield2007/`. sudokucolourHarfield2007.

Tom Hinkle. GNOME Sudoku. Online, Sep. 2006. URL `http://gnome-sudoku.sourceforge.net/`. [Web Site for Computer Program].

Karl King. Sudoku. Online, 2006. URL `http://empublic.dcs.warwick.ac.uk/projects/sudokuKing2006/`. sudokuKing2006.

Simon Yung. SCICS. *Warwick EM Project Archive*, Feb. 1995. URL `http://empublic.dcs.warwick.ac.uk/projects/scicsYung1995/`. scicsYung1995.