

A Show-Me-By-Example Approach to Teaching Programming the Internet-of-Things in Immersive Education

Jeannette Chin

Computing and Technology
Anglia Ruskin University
Cambridge, United Kingdom
jeannette.chin@anglia.ac.uk

Vic Callaghan

Computer Science and Electronic Engineering
University of Essex
Colchester, United Kingdom
vic@essex.ac.uk

Abstract

In this paper we address the issue of how students can learn computer programming using a more natural approach that takes inspiration from an intuitive way people learn and of accomplish tasks (by example), a technical implementation of that approach (programming-by-example) and an immersive educational environment (the eDesk) that combine to provide a novel and accessible approach for students wishing to learn the fundamentals of computer programming.

1.0 Introduction

Learning to read and write programs is a fundamental skill in computing. However, many students struggle to master the basics involved. New networked based technologies, such as the Internet-of-Things are particularly challenging as they are effectively distributed computing systems. The graphical nature of immersive educational environments offers an opportunity to approach learning this type of computer programming in a fundamentally different way. There has been some previous work on developing graphical programming tools that capture spacial and temporal relationships and ease the process of learning to program. For example MIT's Scratch is a well-known teaching tool that captures the essence of "programming" in a way that simplifies the learning process and allows users, in particular young people or children, to creatively construct their own programmed animated sequences via a graphical interface while at the same time, learning the important concepts of programming but without needing them to write any programming code [Malony 2004]. Likewise, Carnegie Mellon's Alice, aims at creating a 3-D programming environment for its users to learn fundamental programming concepts by creating "programs" that interact with the 3-D virtual environment [Kellecher 2006]. In this paper we look at another novel approach to programming called Pervasive Interactive Programming (PiP) that was originally developed for programming intelligent environments, such as smart homes. PiP is part of a larger programming family that is more generically

referred to as “Programming-by-example”, which was introduced by Smith in the mid-seventies, and is based on the principle that required computational functions are converted to executable code through the user demonstrating required behaviour rather than trying to specify it using various abstractions, such as programming languages [Smith77]. PiP differs to Scratch and Alice in that it targets distributed computer systems, such as the Internet-of-Things.


ADDITIVE TECHNOLOGY ePOD-4

In this increasingly competitive world, where knowledge determines success, your child deserves the very best education available and that is Addictive Technology's **ePOD-4**

Pioneering research by Benjamin S. Bloom in the 1980s (and supported by all work since) proved that students who receive one-on-one tuition learn at least an order of magnitude better than grouped students. If you want to give your child the best one-to-one education in the world, give them an Addictive Technology's **ePOD-4**

Education:

- Super-Intelligent Artificial Teachers
- Personalised one-to-one tuition (the gold standard)
- Teacher's avatar has visualisation powers that don't exist in physical space available 24 hours a day, 365 days a year
- Learning environment (avatar, surroundings, lessons) can be tailored for each student
- Unwavering attention and happy disposition
- Compelling content combined with contextual delivery
- Teachers available in different cultures, ages, sexes and form



Technology

- * **FREE-WILL 3** © - Quantum processor (upgradable)
- * **MY-MIND 1.2** © - Evolving Persona Engine (customizable)
- * **FLAME 5** © - EmotionWare
- * **GET REAL B.2** © - Mixed Reality Cocoon
- * **REAL-TOUCH** © iSkin & Haptics
- * **GHOST 4.1** © - 3D Imaging & Audio
- * **SENTINET** © - Knowledge Engine

Addictive Technology, Zizhu Science Park, No. 880 Zi Xing Road, Minhang, Shanghai 200241, China

Figure 1 – Advertisement of ePod from “Tales from a Pod”

2.0 The Immersive Learning Environment – the eDesk

In the paper “*Tales From A Pod*” we described a futuristic student-learning environment called the educational-pod (ePod) [Callaghan 2010]. In Callaghan’s paper, the ePod was described as a small immersive reality cocoon (see figure 1) in which individual students entered to find themselves in a graphically based interactive environment that feigned a real classroom.



Figure 2 – Practical Realisation of the ePod, the eDesk

The environment featured numerous high-tech functions including intelligent tutors in the form of avatars that provided teaching and learning services to the student. Later, a UK company specializing in the manufacture of virtual reality

environments, Immersive Displays Ltd¹, took this vision and translated it into a practical unit, the ImmersaStation², which is the host immersive-learning environment of the work presented in this paper, see figure 2. The concept of the ePod features in two other projects, the Shanghai Jiao Tong Network Education College work on embodying their intelligent student answer machine as an Avatar [Zhang 2011] and Jeddha's King Abduaziz University ScaleUp work on immersive-mixed reality learning [Pena-Rios 2012]. The ImmersaStation runs a virtual classroom called MiRTLE (Mixed Reality Teaching and Learning Environment) [Gardner 2010], which provides a virtual classroom in which lessons are situated (shown in the screen of figure 2). This then forms the immersive learning environment that our novel approach to teaching computer programming targets.

3.0 The Internet-of-Things

Computer networks are becoming ever more pervasive resulting in a world where almost every *“thing”* can be connected to the Internet; the so-called *“Internet-of-Things”* (IoT). In this connected world the balance between the real and virtual is altered, with our daily interactions being split between the physical and electronic domains. For example, the Internet can be embedded into things ranging from bathroom scales through cookers to cars. There are no reliable estimates for the size of this market but a report by the “Arthur D. Little management consultancy” suggests that by 2020 the IoT market could be worth between 22 billion and 50 billion dollars [Schlautmann et al 2011] made up of some 16 billion connected devices [Vermesan & Friess 2011]. This rising demand for Internet-ready *“things”* is good news for graduates as there will be increasing job opportunities for them. However, this new technology demands the acquisition of difficult skills such as programming distributed embedded-computers that, in turn, raise the challenge to educators as to how best to teach these skills. In this paper we present one potential solution to this challenge; Pervasive interactive Programming that we will introduce in the following section.

4.0 Pervasive Interactive Programming

End-User programming is characterised by the use of techniques that allow the end users of an application to create “programs” themselves, without needing to write any code [Cypher et al]. A common way to achieve this goal is to create propriety types of “scripting languages”, abstracting conventional programming algorithms into some form of representations (eg graphical objects) and then provide a platform for the users to manipulate these representations as the basis of learning how to create a program. Most, if not all of existing work on end-user programming targets the development of programs for a single centralised machine, creating programs by manipulating abstract graphical objects.

¹ <http://www.immersivedisplay.co.uk/>

² <http://www.immersivedisplay.co.uk/immersastation.php>

Pervasive-interactive-Programming (PiP) however is aimed at programming distributed computers, embedded into real physical appliances, such as those that make up the Internet-of-Things, which was described earlier. The concept underlying PiP is simple in that it mimics the traditional 'playful' method that has been used to teach children for generations – the teacher demonstrates an action by showing an example; the learner then repeats the demonstrated action.

In PiP, the 'things' that comprise the programming environment are categorised into 2 types: (a) physical 'things' (including graphical representations of them) which we call "hard things", (b) abstract things (eg application software such as email, instant messaging etc,) which we call "soft things". Both types of 'thing' provide their functionalities in a form of services that are network discoverable and accessible.

The availability of numerous networked 'things' and their services present an opportunity to group them together to provide meta-services or meta-appliances. In PiP we called this the "deconstructed appliance or application model" which can be regarded as a form of virtual application / appliance [Chin et al 2006]. The representation (specification) of such virtual entities is referred to as a MAp (Meta-Appliance/Application). It contains detailed information about the community of "things" and rules that govern their functionalities. Rules are created by the end user as part of programming the system.

In our immersive education environment the student is presented with a stimulation of the lab environment that they use to show an example of the required functionality to (which the system encodes as a set of rules). This is also what we sometimes called "*natural programming*". The user is then able to "play" the program and watch the animation to be sure it works as they intended. In this way PiP encourages user interaction as a way that helps the student visualise the programming concepts. In this way PiP acts as a computer programming learning tool, which uses a graphical interface that sits on a virtual reality environment.

5.0 Pervasive Interactive Programming as a Teaching Tool

The inspiration for using Pervasive-interactive-Programming to teach students how to program computers arose from two perspectives. First was the observation that much of the learning in early childhood arises from replicating examples of behaviour that people observe in each other [Brown 2008] and secondly, that at the heart of pervasive interactive programme (a methodology based learning-by-example) are IF-Then-Else rules, that are the core construct of procedural programming languages [Chin et al 2006]. Thus, by providing a mechanism to translate behaviour examples into rules, we have a natural and intuitive model of procedural programming. In the following sections, we explain in more detail the reasoning behind this approach.

5.1 People as a program emulation

An approach that has been adopted by some teachers introducing students to programming, is to mimic the workings of a computational machine by using students to act as the various components in a processors [Kacmarcik 2010]. Thus, for example, a number of students might act as instruction stores, another might be a program counter, another might be an instruction fetch unit, another might be an arithmetic execution unit etc. Then by acting out the sequence of fetching and execution of instructions, students have a visual physical example of how a computer functions. There is something very intuitive in demonstrating or observing physical analogies of abstract processes; a feature that we have tried to capture in our use of PiP as a teaching tool, as the next section will explain.

5.2 PiP as a program emulation

The basic principle of Pervasive Interactive Programming is to capture the macro behaviour of a system as a series of micro tasks, by the user demonstrating to the system examples of the required functionality. PiP captures and describes these micro tasks by means of rules. Sets of these rules are then are combined to form the macro level behaviour of the system. In PiP, rules are normally internalised by the system, and are not usually visible to the user. However, for using PiP as a teaching tool, making the rules visible becomes an important aspect of the pedagogical process. This is important because the core construct of both PiP behaviours and procedural programs are rules. This is illustrated in Figure 3. Rules enable decision-making and decision-making is a key property for any entity that purports to be smart. Its contribution to making Von-Neumann style computers the powerful and flexible problems solving machines we are familiar with, is hugely attributable to there being a decision making mechanism embedded into its structure.

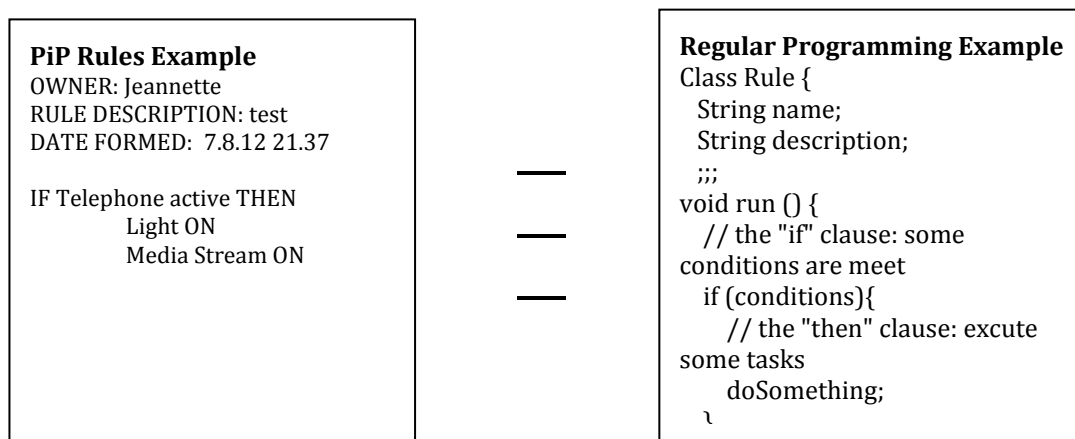


Figure 3 - Equivalence between PiP and Programming decision constructs

It would be hard to exaggerate the importance of decision-making functionality in computation. Without the ability to make decisions computers would be relegated to machines that stepped unidirectionally through lists of instructions. Computers would still be programmed, in the sense someone wrote out those lists of instructions, but the ability of a computer to restructure its computational

strategy and adapt to new problems or contexts would be severely (perhaps fatally) restricted. The use of a decision making structures comes at a large cost; understanding, designing and handling the numerous combinations of resulting program control flows, which has the potential to become a task of daunting complexity. In fact, this is such a big issue it is one of the key targets of software engineering [Callaghan 1982]. Thus for students learning programming a key and complex, aspect is to understand the nature of decision constructs (their relationship to sense-action pairs from data and physical domains), how such decision constructs are formed and how they contribute to the functionality of the overall program. Our approach is to focus on exposing and understanding the role of rule formation as the key construct in computer programming. We do this by employing PiP to show the linkage between sense-action pairs in the real world, and the creation of corresponding rules in the computational machine. We then link these rules to the if-then-else programming constructs in procedural programming languages, which we then use as the launch-pad for students entering and following the more traditional route to learning to program. In this sense our technique is to provide an intuitive and interactive (constructionist) pedagogy for introducing programming concepts to students who are not, initially, technically literate.

5.3A distributed computing programming example

To illustrate the principle of how we use PiP as a tool for teaching computer programming, we will first present a scenario that involves programming a community of 'things' to provide some coordinated functionality. In this example there are three main components being controlled; a telephone, light and media player. The basic idea is that the system should be programmed to provide the behaviour such that when an incoming call arrives, the lights would raise and the media player would stop. Figure 4 – 6 illustrates PiP in use. In a typical teaching session, before logging the system the student is provided with examples of regular programs (eg C and Java) that have the important constructs such as rules, highlighted. At this point they are not expected to understand the program, just to appreciate the main elements. The students are also shown simple examples of programmed coordinated activity (eg a media player coordinating actions with a light). The student is asked to write down the behaviour of the system they wish to create, as a simple state-action list. The student then logs into the system and begin to translate the specification they written into "programs" using PiP interface – via steps (1) exploring things (2) selecting things by dragging and dropping them into the 'programming area' via the graphical interface control panel. In the second phase the student then demonstrates the actions (ie the specified behaviour), which PiP translates into a set of rules, see figure 5. Once the student finishes their "programming exercises" they will able to see the program descriptions in the form of plain text (showing the sense-actions) or actual programming code (the PiP translation comprising rules). The student is then encouraged to replay the program to verify the intended behaviour occurs (ie that the specification is met) before saving their program, see Figure 6. In the third phase the student is invited to look at the constructed rules and relate them to the actions. It is then explained to the

student that a simple procedural language uses these rules as the core decision construct.

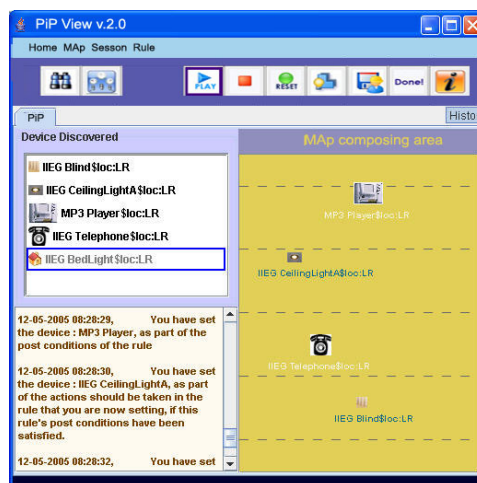
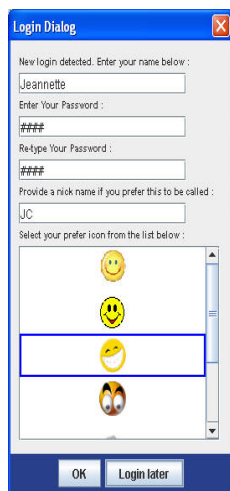


Figure 4 - a) User log in screen b) select 'things' by dragging to composing area

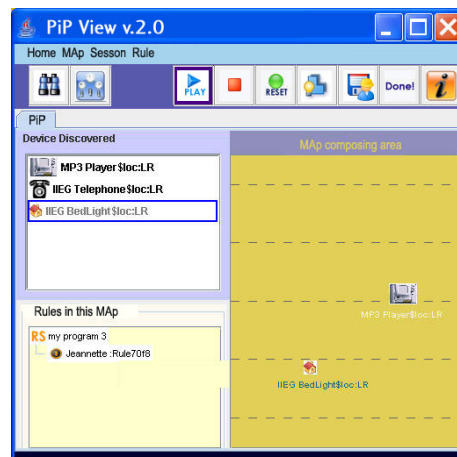
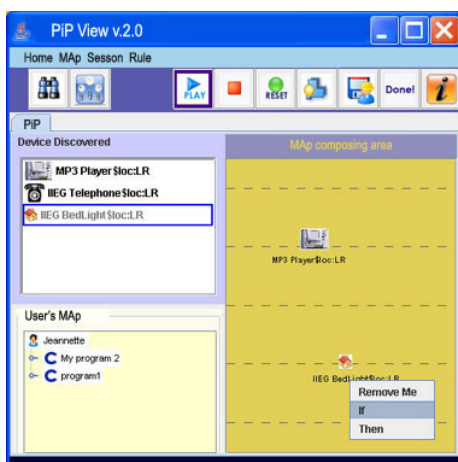


Figure 5 - a) constructing rule b) rule formed (Jeannette_Rule708)

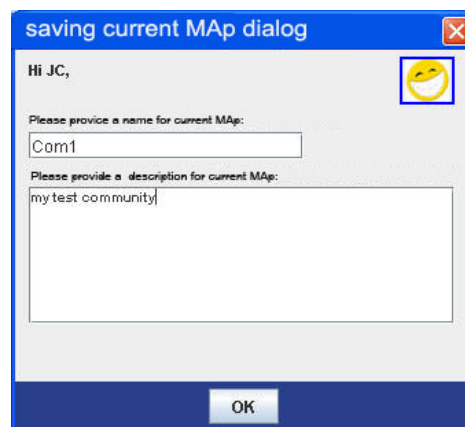
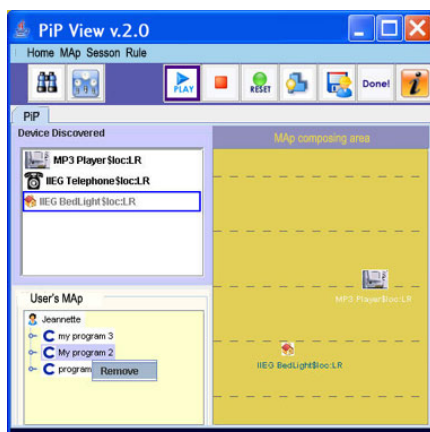


Figure 6 - a) deleting MMaps (rules) b) saving MMaps (rules)

The student is then invited to re-execute the program to verify it's action before being invited to alter the rules manually via the interface, without going through the demonstration cycle again, to make the environment achieve a modified behaviour. The student is then shown how such rules can be translated into actual programming code (by the addition of supporting code such as declarations and operators). Finally, students are then encouraged to compare the manual and automatically generated code to deepen their understanding. Thus, this forms the basic introduction to programming and the task may be progressively made more complex to increase the confidence and programming skills of the student. In our prototype (figures 4-6) we are still experimenting with a 2D view but our next step is to take better advantage of the 3D view that our MiRTLE immersive application facilitates to provide a better immersive experience.

6.0 Summary

In this paper we have introduced a novel end-user programming environment called Pervasive-interactive-Programming (PiP) which we contend offers a particularly intuitive and simple way to introduce new students, or those not majoring in computer science, to programming. The tool goes beyond existing teaching tools, such as Scratch, in that it introduces students to the basics of distributed programming, an intrinsic feature of the newer computing paradigms such as the Internet-of-Things, Pervasive Computing and Intelligent environments etc. We advocate this methodology for immersive education, as it is inherently compatible with the graphical nature of immersive environments. Concerning future directions for our work, there are many possibilities. Our current trials of the system have concerned only small numbers of people, which have allowed us to prove the concept, but we like to expand and formalise this. Also, although we have not exploited this in our current work, the ability of immersive reality environments to materialise abstract programming concepts, such as data structures, semaphores etc offers much potential for the development of the tool in a way that capitalises more fully on the immersive reality nature of the environment we have created. Apart from moving from our current 2D (proof of concept) to a 3D representation, another area we would like to advance our work is to move from what is essentially a single user system to a co-creative team based programming environment. Thus there remain many opportunities for our research to develop which we hope to be able to report in later papers.

Acknowledgement:

We are pleased to acknowledge our colleague Anasol Pena-Rios whose work on end-user programming applications for mixed- reality learning which both motivates and complements this research.

References

- [Brown 2008] Ann L Brown “*Preschool children can learn to transfer: Learning to learn and learning from example*”, *Cognitive Psychology*, Volume 20, Issue 4, October 1988, Pages 493–523
- [Callaghan 2010] Victor Callaghan “*Tales from a Pod*”, Creative Science 2010, Kuala Lumpur, Malaysia, 19-21 July, 2010
- [Callaghan 1982] Callaghan V, Barker K, “*SAS-an experimental tool for dynamic program structure acquisition and analysis*”, *Journal of Microcomputer Applications*, vol. 5 issue 3 July, 1982. p. 209-223, Elsevier Science, ISSN: 0745-7138
- [Chin et al 2006] Chin J, Callaghan V, Clarke G, “*An End User Tool for Customising Personal Spaces in Ubiquitous Environments*”, IEEE the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC-06), Wuhan and Three Gorges, China, 3-6 September 2006
- [Cypher 93] Cypher A, Halbert DC, Kurlander D, Lieberman H, Maulsby D, Myers BA, and Turransky A, “*Watch What I Do: Programming by Demonstration*” The MIT Press, Cambridge, Massachusetts, London, England 1993
- [Kacmarcik 2010] Gary Kacmarcik (Google Inc) “*How Computers Work*”, cse4k12.org, 2010 (http://cse4k12.org/how_computers_work/index.html)
- [Kelleher 2006] Kelleher, C. and R. Pausch. “*Lessons Learned from Designing a Programming System to Support Middle School Girls Creating Animated Stories*”. 2006 IEEE Symposium on Visual Languages and Human-Centric Computing
- [Malony 2004] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. (2004). “*Scratch: A Sneak Preview*”, Second International Conference on Creating, Connecting, and Collaborating through Computing. Kyoto, Japan, pp. 104-109. (<http://llk.media.mit.edu/projects/scratch/ScratchSneakPreview.pdf>)
- [Pena-Rios 2012] Anasol Peña-Ríos, Vic Callaghan, Michael Gardner, Mohammed J. Alhaddad “*Towards the Next Generation Learning Environments: An InterReality Learning Portal and Model*”, The 8th International Conference on Intelligent Environments - IE'12, Guanajuato, Mexico, 26-29 June 2012
- [Schlautmann 2011] Schlautmann A, Levy D, Keeping S, Pankert G, (2011) “*Wanted: Smart market-makers for the Internet of Things*”, Arthur D. Little, management consultancy, see http://www.adlittle.se/prism_se.html?&view=383 (February 2012)
- [Smith 77] Smith, D. C., “*Pygmalion: A Computer Program to Model and Stimulate Creative Thought*”, Basel, Stuttgart, Birkhauser Verlag. 1977.
- [Vermesan 2011] Vermesan O, Friess P, (2011), “*Internet of Things - Global Technological and Societal Trends*” Smart Environments and

Presented at the 2nd European Immersive Education Summit, 26th and 27th November 2012, École nationale supérieure des Arts Décoratifs, Paris, France.

[Zhang 2012] *Spaces to Green ICT*, River Publishers, Denmark, ISBN-10: 8792329675
Tongzhen Zhang, Vic Callaghan, Ruimin Shen ,and Marc Davies "*Virtual Classrooms: Making the Invisible, Visible*" (Presentation), Intelligent Campus 2011 (iC'11), Nottingham 26th July 2011