Vernacular Languages for Mechatronic Making

Alan F. Blackwell
Computer Laboratory
University of Cambridge
afb21@cam.ac.uk

Abstract
This paper proposes a way of thinking about physical making, from the perspective of end-user programming. It addresses current interest arising from the Maker/Hackerspace movement, which aims to democratise the means of creating new mechanical and electronic (mechatronic) devices. By applying insights from the domain of end-user programming, it proposes ways that mechatronic making can be made more accessible to a wider audience, just as programming has been made more accessible to a wider audience through end-user programming.

1. Introduction
There are many fashionable areas of Ubiquitous Computing – Physical computing, Mechatronics, and the Internet of Things – based on the presumption that digital infrastructure can be readily interfaced with the mechanical properties and behaviours of physical objects. However, the practicalities of those mechanical interfaces often require specialist tools and training – electronics, breadboards, soldering irons, workshop tools and so on. Should we assume that specialist expertise is unavoidable in this field, or can this kind of functionality be made accessible to a wider audience?

In the domain of software, we are familiar with the concepts of end-user programming, end-user customisation or end-user development, in which we provide tools usable by people without specialist technical training, that are nevertheless suitable for accomplishing real-world tasks with computers. The purpose of this paper is to explore whether similar principles might be extended to the electro-mechanical systems that are controlled by computers. The initial argument is by analogy between programming (or "software construction") and electro-mechanical design ("physical construction").

In addition to helping people achieve real-world tasks (for example, as supported by spreadsheets), end-user programming (EUP) research is often associated with educational initiatives that aim to broaden access to computing, for example through initial learning environments (ILEs) (McKay & Kölling 2012) such as Scratch, SonicPi or GreenFoot. However, the relationship between end-user programming and educational computing is a complex one (Blackwell 2006a), and it is therefore useful to review a few recurrent issues that often get tangled together here.

1. Both EUP tools and ILEs are designed to be simple and approachable, emphasising a 'gentle slope' from initial experiences to greater computational power (Pane & Myers 2006).
2. The ultimate goal of an ILE is to communicate computational concepts rather than achieve a practical objective, where these might be either broad cognitive skills such as 'computational thinking' (Wing 2008), or more concrete curriculum elements (Computing at School 2012).
3. Nevertheless, many ILEs place a strong emphasis on motivating children to learn programming. This is done by providing EUP support for the creative leisure activities of children, such as story animations (Kelleher & Pausch 2007), video games (Resnick et al 2009), or electronic dance music (Aaron et al in press).
4. When powerful computing capabilities are simplified to a degree that makes them suitable for children, the increased usability is often appreciated by users of all ages – most famously in the outcome of Kay's KiddiKomp/DynaBook project (1972) that led eventually to the development of Smalltalk and the Graphical User Interface.
5. As a result, the design of purely *educational* programming languages such as Pascal or Scratch, even where they are motivated by curriculum rather than application, does often influence the design of more accessible mainstream or general purpose programming languages for practical and end-user applications. For this reason, it is valuable to consider the needs of children, even when research (such as this paper) is aimed at end-users generally.

This paper anticipates that the same dynamics observed in the interaction between EUP tools and ILEs might also apply in the domain of mechatronic construction. To summarise the resulting research questions: Can we identify a 'gentle slope' for mechatronic construction? Can this be based on a more general cognitive understanding? Are there creative applications that would appeal to children? Might the resulting designs lead to more usable and utilitarian tools for everyone?

2. An agenda for end-user construction: Utility, creativity or education?

In developing an analogy between end-user software construction and physical construction, we should note that there are already some physical products that are constructed by end-users, for example self-assembly furniture. In this kind of DIY physical construction market, the tools, components and instructions necessary to accomplish useful practical tasks have been simplified and refined over many years, to a stage where large numbers of people are able to accomplish practical mechanical results without any specialised training.

An interesting consequence of end-user construction, in the case of furniture, is that the end-user places more value in the result, because of the effort that they have invested in it (Norton et al 2012). Although it might appear that there is little 'creativity' in assembling a piece of flat-packed furniture, it appears that the craft experience of working with one's hands is itself a source of creative reward. The craft element of this 'IKEA effect' reinforces the potential value in paying attention to the creative play of children as a source of design inspiration. It also promises potential benefits arising from recent attention to craft aspects of programming (Woolford et al 2010, Blackwell & Aaron 2015).

However in extending the physical construction analogy to creative and educational play, we must consider construction toys such as Lego and Meccano. These toys are even simpler to use than the components of self-assembly furniture, but this is associated with a trade-off. In general, Lego constructions do not have the mechanical strength, motive power, or external 'interfaces' that would be necessary to achieve useful mechanical functions. Furthermore, they are *closed systems* – it is difficult to integrate a Lego construction with an arbitrary piece of woodwork or a cardboard model. In this respect, Lego resembles digital sandbox games such as Minecraft, or even ILEs such as Scratch - it allows you to build toys, and only toys. The reason that so many Lego Technic constructions are vehicles and robots is that this is the only thing it is good for – things that drive around by themselves, but never get attached, mounted or tethered to any part of the child's real life.

The physical interface technologies used in these 'sandboxed' products are not solely an incidental consequence of their original purpose as toys, but also determine their potential future applications. Although it is fun to balance blocks, and snap pieces together with plastic studs or magnets, we do not build our houses, cars or furniture out of these things. Magnets and Lego studs are intentionally impermanent, encouraging reuse, fluidity and experimentation. In terms of cognitive dimensions and their tangible correlates (Edge and Blackwell 2006), they have low viscosity, low premature commitment, high shakiness, and low permanence. They are characteristic of a sketching medium, intended for exploratory design, rather than a construction medium. Furthermore, their uniformity and predictability, although encouraging ease of manipulation, results in a relatively impoverished materiality – their surfaces and forms are smooth and uniform rather than richly textured and resistant. Rather than offering a back-and-forth 'conversation with materials' (Schön 1983) they meekly submit to the user's choice from among a restricted set of possibilities.

In the domain of adult 'end-user' material construction, it is well understood that there is a clear distinction between the physical techniques appropriate to playing with toys, and the physical techniques appropriate to the construction and maintenance of houses, cars, bicycles or shirts. Specialist shops sell the tools and materials required for such serious work, and school pupils are instructed in the special methods necessary for working with such 'resistant materials' (AQA 2012). There may be some basic level of manual dexterity that is transferred from toys such as Lego to screwdrivers and handsaws, but it seems more likely that any correlation is a matter of cultural expectation and self-efficacy, rather than a lowest-common denominator 'mechanical thinking' that could be explained by analogy to the computational thinking movement.

The cognitive and tangible dimensions trade-offs between the design attributes that are associated with playful simplicity, and those that support powerful general-purpose application, is perfectly familiar in the case of software. Neither EUP tools such as spreadsheets, nor ILE game authoring tools such as Scratch, have the versatility or computational power of general purpose programming languages. Much programming language design, whether for end-user developers or children, involves identifying a sweet-spot between power and usability, as well as a gentle slope for those users who become more familiar with the tools, more ambitious in their goals, and want to explore a wider range of functionality. Given the central role of social expectation and self-efficacy in both domains, it would appear valuable to address these issues together, spanning both mechanical and computational ways of thinking. Indeed, a previous PPIG paper exploring the development of new creative tools for the Internet of Things evaluated the self-efficacy of professional artists as potential end-users by drawing an analogy from their familiarity with mechanical construction tools to their confidence in first encountering programming (Blackwell, Aaron & Drury 2014).

3. Why a vernacular language?

The agenda set out in the previous section offers a motivation for a new kind of 'end-user construction language' for mechatronics, that is distinct from sandboxed toys like Lego, and could potentially be informed by past research in end-user programming language design. To be clear, this is taking a psychology of programming approach, but applying it to questions that are *not* normally related to programming. It is not usual to speak of physical objects or electronic circuits as if they were a 'language,' other than through the structural semiotic perspectives of cultural studies as developed by Roland Barthes and others. My goal here is more abstract, proposing an engineering language (rather than a cultural one), but one that can be used to describe physical objects and mechanisms from a functional perspective that is accessible to, and usable by, end-user audiences other than engineers.

I should note that although the emphasis of this paper is to investigate physical construction rather than software, it may well be the case that programming would also be involved in an actual end-user project enabled by such a language, for example if novel mechanical devices were being attached to microcontrollers or embedded devices such as Raspberry Pi, Arduino, ARM mBed, and many others. This is an obvious (and likely) extension of the discussion in this paper, but will not be addressed further here, in order to concentrate on the novel aspects of the proposal.

3.1. Functional description as API specification

The argument so far has taken a determinedly user-centric perspective, in setting out the case for an end-user construction language by analogy to end-user programming. Nevertheless, this analogy is also wholly plausible from a technical perspective. In systems design, it is well understood that information flows between software and the physical world via sensors and actuators, and that system functionality is embedded in and constrained by physical apparatus such as cases and mountings.

Although sensors, actuators, I/O peripherals, and the physical environment of the computer are usually considered to be outside the scope of programming language design, it is possible to analyse their functional descriptions as a more extensive and general kind of Application Programming Interface (API). Just as APIs can be designed to offer a more consistent set of user-centric abstractions with the assistance of Cognitive Dimensions analysis (Clarke 2003), a language for electro-mechanical abstraction might also be constructed from a user-centric perspective. We might even describe this as a Cognitive Dimensions of Physics, in which the physical world is treated as a kind of notation, through the functional language that we use to analyse and describe it.

3.2. Naturalness in API descriptions

If we apply this user-centric strategy to evaluate the I/O interfaces and mounting hardware that are conventionally used for mechatronic projects, this offers an interesting critical perspective, by analogy to the design of the most commonly used APIs: those of language utility libraries. General purpose programming languages such as Java (in contrast to end-user languages) often rely heavily on libraries that implement generic computational abstractions – lists, stacks, heaps and so on. Without prior education in the algorithms and data structures of computer science, those abstractions are rather impenetrable to the end-user.

Domain-specific libraries tend to be described in terms that are more directly related to the user's actual tasks, but it is often the case that such libraries build on the intermediate abstractions of general purpose utilities, making them also difficult to use and understand for those unfamiliar with the basic abstractions. Designers of conventional programming languages are often unable to accept that this is problem, or even to notice it. From their expert perspective, conventional library abstractions such as a stack have become 'natural' or even 'intuitive,' because of the way they are so tightly integrated with the design of mainstream programming languages. As a result, new libraries and APIs continue to be designed according to the same abstractions as those that are already established. If there is a problem, experts often perceive this as a problem of education, to be addressed by imparting the computational thinking principles that are apparently necessary to use conventional abstractions.

In contrast, some end-user programming systems extend the principles of domain-specific language design to deeper levels of abstraction, by ensuring that the semantics and syntax of the language are derived from abstractions and notations already familiar to the user. This is relatively routine in the field, as used for example in the "Natural Programming" project of Myers et al (2004), in which educational programming languages such as Pane's HANDS (2002) were derived from the vocabulary and concepts that children already used to describe behaviour in videogames.

3.3. Naturalness and naïve physics

An interesting liaison between applied anthropology and artificial intelligence during the 1980's (Gentner & Stevens 1983) encouraged the field of *naïve physics* – an attempt to build computer systems that could reason about physical phenomena following common-sense principles, rather than those of trained physicists (Hayes 1978). The goal of such research was in part to simulate human intelligence, but also to produce expert systems that might be able to explain the reasons for their decisions, in terms that were accessible to lay people. The legacy of naïve physics includes qualitative spatial reasoning systems (Forbus 1983), which support navigation or spatial queries expressed in natural language rather than geometric terms.

My purely-visual language *Palimpsest* (Blackwell 2014) consciously applied this perspective to programming language design. Having previously conducted research in the field of qualitative spatial reasoning (Blackwell 1989), it was apparent that Myers' Natural Programming approach could be applied to non-verbal domains such as image processing, using qualitative terminology. Of particular interest was the development of a set of data types and geometric operations that would be adequate for a wide range of pictorial transformations and processes. Many of the Palimpsest functions were initially implemented in terms that would be familiar from school geometry, but subsequently re-implemented using concepts that offered more "natural" interaction with images. For example, the mathematical operation "translate" was replaced with "move it to here". Similarly, the most common application of "rotate" turned out to be "make it spin round" (as an animation), and "scale" was "stretch or squash" (not uniformly, but in various ways that drag handles can produce). Similarly, the basic data types of Palimpsest were refined to represent the most salient features of images: basic data types include 'image', 'colour', 'shape' and 'location' rather than conventional mathematical data types such as 'integer' and 'real'.

As was the case with expert systems employing naïve physics, geometric processes defined in terms of naïve computation are unlikely to be as concise or generalizable as conventional mathematical terms and notation. However, this trade-off applies to many innovations in HCI (the GUI versus the command line, for example). To some extent, user-centred design always involves stepping away from an existing, possibly elegant, engineering description in order to accommodate alternative mental models. From this perspective, the goal of the 1980s naive physics movement could have represented a user-orientation within the AI community, trying to create knowledge representations that were better aligned with common sense although somewhat inelegant. The anthropological origins of the "naïve" movement always emphasized the need to respect the customs and reasoning strategies of other communities, rather than simply disparaging them as undisciplined or murky thinking.

3.4. From natural to vernacular

Although the ambitions in applying Myers' Natural Programming project to EUP (Pane & Myers 2006) are laudable, the potential for broadly differing interpretations of 'naturalness' means that it is not always clear how this ambition might be carried through into language design. What is natural to one person (a computer scientist) is not necessarily natural to another (a child), and indeed something that seems natural to a person at one age may not seem natural to the same person at another time. Furthermore, to return to the cautions given in the first section of this paper, the distinction between EUP environments and ILEs is not always clear. In some cases, ILEs are designed with the intention of communicating curriculum concepts, and thus *changing* the 'natural' (i.e. uneducated) conceptions that the users might previously have held (Rode et al 2003). In other cases, domain-specific languages emphasise the use of terminology and concepts that are already understood by users. When the users are children, the problem arises whether their 'natural' terminology should be regarded as arising from domain expertise, or from lack of understanding.

The perspective offered by the naïve physics agenda does sidestep this dichotomy, by clarifying that there may be valid and useful descriptions that do not correspond to textbook definitions. Of course, we should remember that the word 'naïve' is not very diplomatic in a user-centred design context, where it is important to use terminology that respects the user's competence and judgment, rather than accusing them of naivety. The key insight in naïve physics is not ordinary people are naïve, but that they are generally able to act quite successfully in the physical world without recourse to specialist expertise, whether Newtonian mechanics, quantum physics, Ohm's law, or computational thinking.

Rather than 'natural', the goals of this paper seem to be best served by the word 'vernacular' – which is to say, the kind of informal and everyday language that is used by ordinary people in their native context, *as opposed to* the kind of language that is used by professional specialists from other communities. This word makes it clear that naturalness of language or description is not a universal attribute, but rather one that is specific to particular cultures and groups of people. In particular, we should not assume that vernacular descriptions of software should be the same among computer scientists and end-users, and we should not assume that vernacular descriptions of mechatronics should be the same among engineers and children. (Remembering here, point 4 of my opening rubric, which is that designing for children has often served as a useful strategy in escaping previous design fixation – the intention is not simply to achieve a vernacular appropriate only to children, but to use the critical lens of creative childhood experience as a means of reconceptualising end-user tools).

4. Toward playful electro-mechanical abstractions

In order to evaluate the potential benefit of this strategy, this section considers the ways in which current mechatronic construction technologies demonstrate the properties discussed above, and then suggests some initial features of an alternative vernacular language. Throughout, I maintain a focus on the potential for supporting creative childhood experience, but as explained, with the intention that this is likely to be a valid strategy for broader future applicability.

4.1. The abstractions of mechatronics

A quick survey of online accessory catalogues for popular hobbyist platforms such as Raspberry Pi, Arduino, Phidgets and .NET Gadgeteer reveals an array of specialist terminology, including 'potentiometers', 'quadrature encoders', 'relays', 'gyroscopes', 'solenoids', 'accelerometers', 'proximity sensors', 'differential air pressure', 'thermocouples', 'linear actuators' and many more. As an engineer, many of these devices are familiar to me, and there must clearly be some demand for them among the customers of such catalogues, otherwise they would not be offered for sale. However, they do not appear at first sight to represent a vernacular language outside of this technical community, and certainly not for the children who I have proposed as a target audience.

If we compare these interface components to the standard APIs of programming language utility libraries, it is clear that there is a specialist conceptual language embedded in the names of the components, and also in the conceptual design of their functionality. The technical terms are precise, but the concepts that they represent are not easily described in a more vernacular manner, other than by mundane explanation.

If we look beyond these relatively well-encapsulated sensor and actuator devices, the situation deteriorates. A typical introductory presentation explaining how to 'Blink an LED' with a Raspberry Pi reads as follows:

> *An LED is a Light Emitting Diode. A diode is a circuit element that allows current to flow in one direction but not the other. Light emitting means ... it emits light. Your typical LED needs current in the range of 10-30 mA and will drop about 2-3 volts. If you connect an LED directly to your Pi's GPIO it will source much more than 30 mA and will probably fry your LED. To prevent this we have to put a resistor. If you want to do math you can calculate the appropriate resistance using the following equation ... [and so on]*
>
> (Minardi 2013)

In discussing the relationship between domain-specific languages and general purpose ones, I introduced the problem of 'intermediate abstractions' that might be considered natural to a person already familiar with a wide range of programming languages, but would not have been encountered by someone who does not have the necessary background in computer science or computational thinking. The 'intermediate abstractions' of these standard electro-mechanical interface components and techniques are similarly foreign to end-users. From current-limiting resistors as described above, to op-amps, transistors and many other components, we find abstractions that are familiar to electronic engineers, but not to DIY handymen, or children (or even computer science undergrads). The mechatronic 'library' components of solenoids and thermocouples, even when packaged for end-user plug-and-play, still require a conceptual engineering language to describe and apply.

Furthermore, while every child can join Lego bricks, many are unable to strip the insulation from a wire, use a soldering iron, read the value code of a resistor and so on. Furthermore, even if we consider the cruder level at which an engineer or hobbyist might mount a Raspberry Pi or Arduino in a wooden box, or shape the coupling between a solenoid and a lever, many children (or adults) might struggle to hammer in a nail, or cut a straight line with a handsaw and file. Although those are routine skills and concepts for an engineer, and even for the typical Raspberry Pi 'maker' hobbyist, they present obstacles for children who might want to emulate those adult examples.

4.2. Physical craft as vernacular

Might it be possible, when identifying and designing user-oriented electro-mechanical components, to specify them, not in standard engineering terms ('solenoid', 'stepper-motor', 'microswitch'), but in terms related to the practical experiences of an imaginative child ('cardboard', 'string' and so on)? Ideally, these should not be constrained to toy assembles and sandbox contexts, but potentially usable for practical tasks, offering a gentle slope from first experiences to useful applications. A reasonable benchmark would be the mechanical properties of Ikea furniture – requiring only a small range of physical construction skills, but sufficiently strong and powerful to support real-world functionality. So although toy makers such as Lego do provide sensor and actuator interfaces to introductory programming languages, the objective would be to avoid these in favour of a craft vernacular that is engaged with adult life.

By analogy to domain-specific programming languages, such components could have electro-mechanical functions that are understandable to children because they correspond to mechanical actions already made by children. Examples include:

- Pull a string
- Squirt a hose
- Swing a door
- Flick a switch

The advantage of household functions, rather than toy scenarios, is that they can be integrated with the existing commercial infrastructure of standardised electro-mechanical components that are familiar to children:

- Strings can be tied to nails or hooks
- Hoses can be connected using the Hozelock system

- Door hinges can be nailed or screwed to pieces of wood
- Switches can power small appliances via extension cords and plugs

Although it is attractive to provide a gentle slope from childhood play to practical application, parents of Lego-using children who read this might balk at the new potential for harm suggested by these scenarios – with power comes responsibility! Nevertheless, each of these interface technologies is also regulated by safety standards specifically intended for domestic use, and there is no reason in principle why the creativity of children should be completely isolated from the adult world.

Alternatively, there are vernacular mechanical actions that can be applied to physical objects, without involving sufficient force for damage, but offering social and communicative potential:

- Pluck
- Twist
- Knock
- Brush
- Tickle
- Scratch
- Feed (a pet)

Finally, while conventional sources of motive power intended for the engineering functions of a typical house are sufficiently powerful to be somewhat dangerous, many natural forms of energy are relatively safe. Relatively small amounts of surplus energy in these scenarios would even be sufficient to power an embedded processor offering end-user programmable functionality. (The last two are somewhat fanciful, but perhaps more appealing to children).

- Wind chime – powered by wind, makes sound
- Grandfather clock – powered by wind-up weight, pulls string
- Light show – powered by daytime sun, makes bright light at night.
- Dog whistle – powered by motion of pet, communicates with pet
- Stink bomb – releases smell, powered by motion of trousers

4.3. Systematising vernacular abstractions

Much of the enthusiasm driving the naïve physics movement may have been associated with the academic impulse to systematise the hitherto unsystematic. It is possible that the same impulse adds appeal to the research agenda of natural programming. It is certainly likely that the abstract functions listed above could be collected more systematically, for example as in the application of Reuleaux's taxonomy of kinematic pairs to analyse the possible configurations of moving parts in a tangible programming language (Blackwell & Edge 2009).

However, it might be wiser to resist this impulse to systematise, especially at this early stage. As observed by Umberto Eco, the desire to systematise vernacular languages is a largely Quixotic enterprise (Eco 1995). There are several sources of systematicity already embedded in the illustrative suggestions made above, which arise in part from:

- The commercial dynamics of industrial standardisation
- The distinction between classes of domestic energy reticulation
- The refinement of building materials to fill economic niches

If new end-user tools were to be developed in response to the arguments presented here, then the necessary engineering design process would itself impose commonalities and distinctions, whether or not they were 'naturally' present in the vernacular contexts. If there are forms of tacit knowledge embedded in craft, that might be resistant or even opposed to that engineering logic, then it would be advisable to follow an explicit strategy of late-binding in future work that develops these ideas. Rather than looking for a mechatronic language that resembles the existing infrastructure of sensors and actuators, while simply substituting alternative names or power sources, there might be more opportunity for insight from paying closer attention to the physical vernacular of everyday life, as in the 'unremarkable computing' of Tolmie et al (2002).

One interesting opportunity is the potential to focus on materials that are softer or more malleable than typical engineering components. At the engineering end of the hacker/maker movement, as in student laboratories, robot competitions and science fair projects, end-user construction is facilitated by products and standards such as the OpenBeam construction system – a kind of larger-scale Meccano. However, as the basis of a vernacular language, systems such as OpenBeam are oriented toward a very characteristic building style – precise, but also hard- (and sharp)-edged. We should perhaps seek opportunities to support soft materials such as fabric, fur, feathers and faces, and vernacular 'interfaces' to clothing and other social signifiers such as zippers or outfits. As I have noted in the past, these types of material support abstraction, while resisting the gender normativity that is associated both with engineering construction and with DIY (Blackwell 2006b).

5. Related Work

The challenges in interfacing low-cost computers for practical mechanical purposes are extremely well known, and there are countless historical and contemporary products that have addressed this potential market. The two most common product classes at present can be grouped under the general headings of robotics kits, and IoT / smart home products. Robotics kits tend to be aimed at children or for classroom use, while IoT hub products are aimed at the hobbyist market. However, there is some degree of crossover between the two markets, with major products such as Arduino and Raspberry Pi catering to both, and a range of accessories and guidebooks oriented toward one or the other.

Most of this activity, and in particular the large number of Kickstarter initiatives that cater to one or other of these markets, is driven by technical enthusiasm and subject to the two primary flaws that I have discussed: the electro-mechanical robots tend to be autonomous in a way that restricts them to toy sandboxes, while the hobbyist applications are modelled on existing engineering descriptions and components. I will not review these in detail, other than to note their prevalence.

There have been a small number of exceptions, generally emerging from academic research contexts. MaKey MaKey (Silver et al 2012) is an interface kit based around the abstraction that any object can be treated as a 'key' (in the sense of a key on the computer keyboard), and can then control software applications in accordance with that abstraction. The digital 'key' abstraction is supported by auto-ranging and signal conditioning of analog inputs, such that electrical connections can be made to a wide range of objects with varying conductive and capacitive characteristics, and in particular, natural objects such as plants, fruit and vegetables.

A series of projects by Orth et al (e.g. 1998) has explored the opportunities for computers to be mounted in, and interfaced with, fabrics of various kinds. Circuits can be constructed using embroidered or woven conductive thread, although mounting rigid chips and controllers has often been problematic (Buechley & Eisenberg 2009). The majority of fabric computing applications have focused on toys, soft furnishings and wearables, with the last of these often featuring LEDs or colour-changing materials that respond to touch and temperature. To date, these have generally been decorative (although this is undeniably a functional feature of wearable technology) rather than delivering end-user capability for novel engineering functionality.

Among the many educational robotics systems, Schweikardt's (2011) modular robotics system 'Cubelets' presented an alternative set of abstractions for sensing and actuation, based on a metaphor relating these peripherals to the capabilities of living organisms rather than engineering systems. The Cubelets kit interestingly includes magnet-to-Lego interface components, but the ingeniously fluid magnet mechanism is not compatible with mechatronic component standards. Furthermore, communication with the external world is achieved only via Bluetooth wireless, making Cubelets impractical for inclusion in larger assemblies.

Microsoft's .NET Gadgeteer (Villar et al 2012) offers a relatively conventional set of hardware peripheral modules, similar to those of the Phidgets system, but offers a software-oriented abstract view through the Visual Studio development environment. Each module is associated with a C# class that is automatically loaded into the development environment when that module is connected to a tethered Gadgeteer master unit. As a result, the functionality of the module can be presented from a software perspective rather than focusing on hardware description.

6. Conclusions

This paper has proposed that the distinctive user-centric perspective characteristic of psychology of programming research could also be applied to the physical context of computing – the construction materials, sensors and actuators that are involved when a computer is embedded in a physical context. At present, educational computing systems tend not to be strongly attached to the physical world, while hobbyist technologies are defined in specialist engineering terms. As an alternative, I have suggested that a vernacular language of mechatronic construction could be used to shape components, standards, products and their capacities for playful, creative and useful end-user applications.

7. Acknowledgements

I am grateful to Robert Mullins, Saar Drimer, Jeff Osborne, Mark Gross, and Nic Villar for conversations that have contributed to the development of ideas in this paper.

References

Aaron, S., Blackwell, A.F. and Burnard, P. (in press). The development of Sonic Pi and its use in educational partnerships: co-creating pedagogies for learning computer programming. To appear in a 'Live Coding in Music Education' special issue of *The Journal of Music Technology and Education*

AQA (Assessment and Qualifications Alliance) (2012). *GCSE Specification: Design and Technology: Resistant Materials*. http://www.aqa.org.uk.

Blackwell, A.F. (1989). Spatial reasoning with a qualitative representation. *Knowledge-Based Systems*, 2(1), 37-45.

Blackwell, A.F. (2006a). Psychological issues in end-user programming. In H. Lieberman, F. Paterno and V. Wulf (Eds.), *End User Development*. Dordrecht: Springer, pp. 9-30

Blackwell, A.F. (2006b). Gender in domestic programming: From bricolage to séances d'essayage. Presentation at *CHI Workshop on End User Software Engineering*

Blackwell, A.F. (2014). Palimpsest: A layered language for exploratory image processing. *Journal of Visual Languages and Computing* **25**(5), 545-571.

Blackwell, A.F. and Edge, D. (2009). Articulating tangible interfaces. In Proc. *Third International Conference on Tangible and Embedded Interaction (TEI'09),* pp. 113-118.

Blackwell, A.F., Aaron, S. and Drury, R. (2014). Exploring creative learning for the internet of things era In B. du Boulay and J. Good (Eds). *Proceedings of the Psychology of Programming Interest Group Annual Conference (PPIG 2014)*, pp. 147-158.

Blackwell, A.F. and Aaron, S. (2015). Craft Practices of Live Coding Language Design. In *Proc. First International Conference on Live Coding*. Zenodo. http://doi.org/10.5281/zenodo.19318

Buechley, L., & Eisenberg, M. (2009). Fabric PCBs, electronic sequins, and socket buttons: techniques for e-textile craft. *Personal and Ubiquitous Computing*, *13*(2), 133-150.

Clarke, S., & Becker, C. (2003). Using the Cognitive Dimensions Framework to evaluate the usability of a class library. In *Proc. First Joint Conference of EASE & PPIG (PPIG 15)*.

Computing at School Working Group (2012). *Computer Science: A curriculum for schools.* Available online at http://www.computingatschool.org.uk

Eco, U. (1995). *The search for the perfect language*. Wiley-Blackwell.

Edge, D. and Blackwell, A.F. (2006). Correlates of the cognitive dimensions for tangible user interface. *Journal of Visual Languages and Computing*, 17(4), 366-394.

Forbus, K.D. (1983). Qualitative reasoning about space and motion. In Dedre Gentner and Albert L. Stevens, editors, *Mental Models*. Lawrence Erlbaum Associates.

Gentner, D. and Stevens, A.L. (Eds) (1983). *Mental Models*. Lawrence Erlbaum Associates.

Hayes, P.J. (1978). The naive physics manifesto. In D. Michie, ed, *Expert Systems in the Micro-Electronic Age*. Edinburgh University Press.

Kay, A. C. (1972). A personal computer for children of all ages. In *Proceedings of the ACM annual conference-Volume 1* (p. 1). ACM.

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, *50*(7), 58-64.

McKay, F., & Kölling, M. (2012). Evaluation of Subject-Specific Heuristics for Initial Learning Environments: A Pilot Study. In *Proc. 24th Psychology of Programming Interest Group (PPIG'12)*.

Minardi, J. (2013). open('dev/real_world'): Raspberry Pi sensor and actuator control. Presentation at *SciPy 2013*. Summary available online at http://blog.enthought.com/general/raspberry-pi-sensor-and-actuator-control/

Myers, B. A., Pane, J. F. and Ko, A. (2004). Natural Programming Languages and Environments. *Communications of the ACM*, 47(9), 47-52.

Norton, M.I., Daniel Mochon, D., Ariely, D. (2012). The IKEA effect: When labor leads to love. *Journal of Consumer Psychology* 22, 453–460

Orth, M., Post, R., & Cooper, E. (1998). Fabric computing interfaces. In *CHI 98 conference Human Factors in Computing Systems,* pp. 331-332.

Pane, J.F., Myers, B.A., & Miller, L.B. (2002). Using HCI techniques to design a more usable programming system. In *Proc. IEEE Symp. Human Centric Computing Languages and Environments,* pp. 198-206.

Pane, J.F., and Myers, B.A. (2006). More natural programming languages and environments. In H. Lieberman, F. Paterno and V. Wulf (Eds.), *End user development*, pp. 31-50.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67.

Rode, J.A., Stringer, M., Toye, E., Simpson, A.R. and Blackwell, A. (2003) Curriculum focused design. In *Proc. ACM Interaction Design and Children*, pp. 119-126.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Basic books.

Schweikardt, E. (2011). Modular robotics studio. In *Proc. fifth Int. Conf. on Tangible, embedded, and embodied interaction (TEI'11)*, pp. 353-356.

Silver, J., Rosenbaum, E. and Shaw, D. (2012). Makey Makey: improvising tangible and nature-based user interfaces. In *Proc. Sixth Int. Conf. on Tangible, Embedded and Embodied Interaction (TEI'12)*, pp. 367-370.

Tolmie, P., Pycock, J., Diggins, T., MacLean, A., & Karsenty, A. (2002, April). Unremarkable computing. In *Proc. SIGCHI conference on Human factors in computing systems*, pp. 399-406.

Villar, N., Scott, J., Hodges, S., Hammil, K., & Miller, C. (2012). NET gadgeteer: a platform for custom devices. In *Proc. Pervasive 2012,* pp. 216-233.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725.

Woolford, K., Blackwell, A.F., Norman, S.J. & Chevalier, C. (2010). Crafting a critical technical practice. *Leonardo* 43(2), 202-203.