

Dependency by definition in Imagine-d Logo: applications and implications

Chris Roe, croe@dcs.warwick.ac.uk

Centre for New Technologies Research in Education, The University of Warwick, Coventry, UK

Meurig Beynon, wmb@dcs.warwick.ac.uk

Dept of Computer Science, The University of Warwick, Coventry, UK

Abstract

Dependency is a concept whose importance for learning is strongly suggested by – amongst other things – the wide range of applications that spreadsheets have found in education. Given the prominent role that Logo has played in research into computer support for constructionist learning, there is a natural motivation for studying the relationship between spreadsheets and Logo programming. The technical issues surrounding the implementation of dependency in Imagine Logo (Kalas & Blaho, 2000) have been addressed in previous work of Peter Tomcsanyi (2003). Tomcsanyi's research gives useful insight to the Imagine Logo programmer, but does not indicate how dependency might be exploited by end-users the high-level way that it has been in spreadsheets. This paper introduces a prototype extension of Imagine Logo, informally called **Imagine-d Logo**, that allows dependencies to be specified using spreadsheet-style definitions.

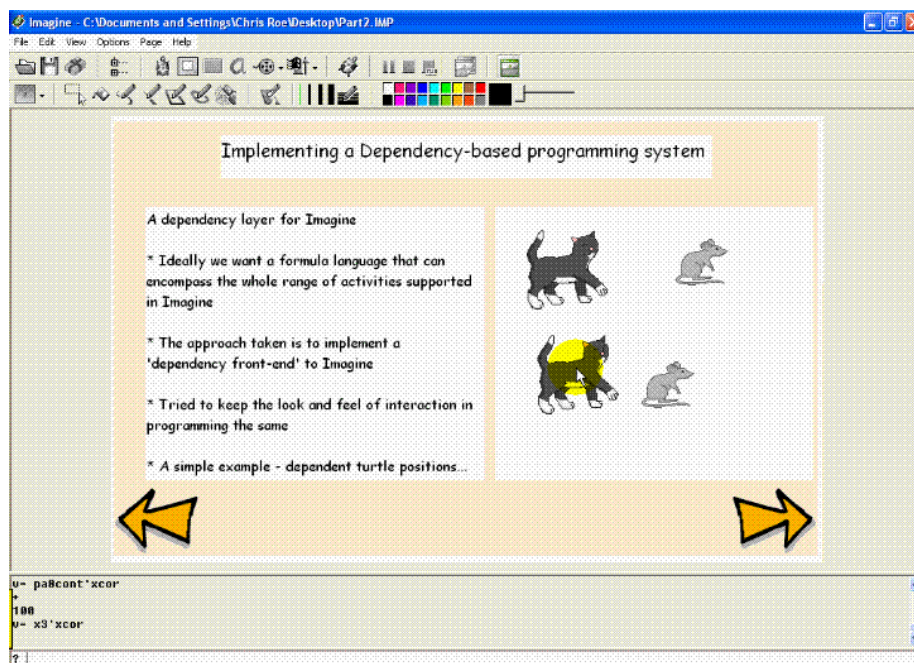


Figure 1. A slide from a presentation developed using Roe's **Imagine-d Logo** prototype

The current scope and potential for Imagine-d Logo are discussed and illustrated with reference to practical issues encountered in using Imagine Logo to program a microworld, to implement spreadsheets and to give computer support for presentations. The broader implications of introducing *dependency by definition* into a Logo environment in this manner are briefly discussed from the perspectives of education and computing.

Keywords

Imagine Logo, dependency, spreadsheets, Empirical Modelling, programming paradigms

Introduction

Spreadsheets and Logo – in all its rich and numerous variants – have been amongst the most influential applications where computing support for education is concerned. By way of background, Baker and Sugden (2003) describes the wide range of educational applications for spreadsheets, and *What is Logo?* (Logo Foundation) gives a good account of the impact of Logo on education. It is natural to seek to integrate these two approaches to educational technology. Such an integration raises challenging issues from several different perspectives:

- **Practical programming:** What motivating advantages might there be in introducing spreadsheet principles into Logo and how is this best accomplished technically?
- **Pedagogy:** What is the significance of spreadsheets and Logo as vehicles for learning, and what is the relationship between them?
- **Computing science:** How can we make unified sense of the two different computational paradigms represented in spreadsheets and Logo?

Our initial and primary emphasis here is on the practical programming perspective. We introduce a prototype extension of Imagine Logo, developed by the first author, that was motivated by issues encountered in programming microworlds based on the throwing events of the Olympic Games (Roe et al, 2005). This prototype - to be referred to as *Imagine-d Logo* - builds on previous work by Tomcsanyi (2003), but more closely resembles Geomland (Sendov & Dicheva, 1988) in its exploitation of dependency. Another implicit influence stems from both authors' work on the principles and tools of Empirical Modelling [EM], a general approach to computing that can be viewed as rooted upon a methodology for modelling with dependency (EM website; Beynon, 2007). The later sections of the paper discuss the Imagine-d Logo prototype from the broader pedagogical and computational perspectives associated with EM thinking, and conclude by considering the potential implications for the future development of Logo-like languages.

Adding spreadsheet-style capabilities to Imagine Logo

The spreadsheet is the archetypal example of an application that exploits *dependency maintenance*, the mechanism whereby changing one value propagates changes to other values in a predictable way. The idea of dependency is often informally invoked by programmers in a much more general context. In implementing object dependencies in Imagine Logo, Tomcsanyi (2003) is concerned with quite general situations in which an object's attributes depend on the attributes of another.

Tomcsanyi's approach to implementing dependencies within an object-oriented programming framework is similar in spirit to that of Perry (2001). Dependencies are introduced by the programmer through embellishing the underlying classes in ways that demand considerable technical skill. By contrast, making computer models using spreadsheets is an activity that can be carried out by non-specialists, and can be viewed as a form of end-user programming (see e.g. Nardi, 1993). In order to introduce dependency into Imagine Logo in a way that gives similar scope for the non-expert programmer, certain key requirements must be met. The programmer:

- should not be concerned with the way in which the underlying system maintains the dependencies between objects in the program;
- should be able to ascertain the definition of an attribute/object/variable by inspection.

To achieve the kind of expressive power that Tomcsanyi (2003) realises in his implementation of object-dependencies, it is also desirable to support a wider range of dependency relations than is supported by a traditional spreadsheet. For instance, whereas the spreadsheet is best suited to maintaining scalar relationships recorded in a tabular format, it is natural to consider maintaining relationships between geometric objects in the Imagine Logo environment. By way of illustration, Figure 2 displays what Tomcsanyi (2003) identifies as "a kind of canonical construction of dynamic geometry" - a triangle together with the common point of intersection of the perpendiculars dropped from its vertices onto the opposite side.

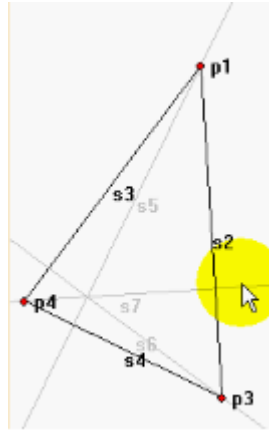


Figure 2. A simple dynamic geometry construction

In this construction, the geometric relationships are maintained by dependency when the position of any of the red vertices p1, p3 and p4 is changed.

Neither of the above requirements is met in Tomcsanyi's implementation of dependency in Imagine Logo. The Imagine-d Logo prototype addresses these requirements by making it possible to specify and revise the dependencies between attributes of objects in much the same way that this is done in a spreadsheet: by formulating and editing explicit definitions for attributes. This has the advantage of making dependency relationships much easier to understand and to trace.

From a technical programming perspective, Roe's extension of Imagine Logo is similar to that described by Tomcsanyi (2003). The same underlying principles are used to maintain dependency, but the details of the implementation are hidden from the programmer. This potentially means that end-user programmers can make effective use of dependency. It also promotes qualities that can be found in modelling with spreadsheets, making models simpler to specify, quicker to build, and more readily open to extension.

A dependency layer for Imagine Logo

The Imagine-d Logo prototype was developed without access to the Imagine Logo source code: it accordingly adds a dependency layer at the dialogue-box level to Imagine Logo. This layer supplies a 'front-end' in which attributes can be given values using definitions that can be specified and edited dynamically in much the same way as the cells of spreadsheet. This involves developing a language within which to specify dependencies between many different kinds of attributes – ideally, a language sufficiently rich to support all aspects of programming activity afforded by Imagine Logo. A further important consideration is that the extension of the interface should respect the look and feel of the Imagine Logo programming environment.

The full technical details of the implementation are beyond the scope of this paper. The features of the prototype will be motivated and illustrated with reference to practical programming issues that were encountered in the process of building the Olympic Games microworlds.

A simple example of the kind of dependency to be specified is that between the positions of two turtles that is discussed in Tomcsanyi (2003). The relationship between the position of the cat and the mouse in Figure 1 illustrates the general idea. In the pane on the right-hand side of this figure dragging movements of the cat result in left-right movements of the mouse that maintain a fixed horizontal distance between the two. The way in which the model-builder specifies this dependency differs between the top and bottom cat-mouse pairs, though the effect is the same.

The specification of the dependency between the positions of the cat and the mouse in the top cat-mouse pair in Figure 1 typifies the way in which a relationship between attributes is expressed in an Imagine Logo program. As displayed in Figure 3, the specification takes the

form of a triggered action – located in the Events field – that is invoked whenever the cat (here implemented by the turtle x1) is dragged about the screen. The effect of this triggered action is to update the x coordinate of the mouse (here represented by the turtle x2).

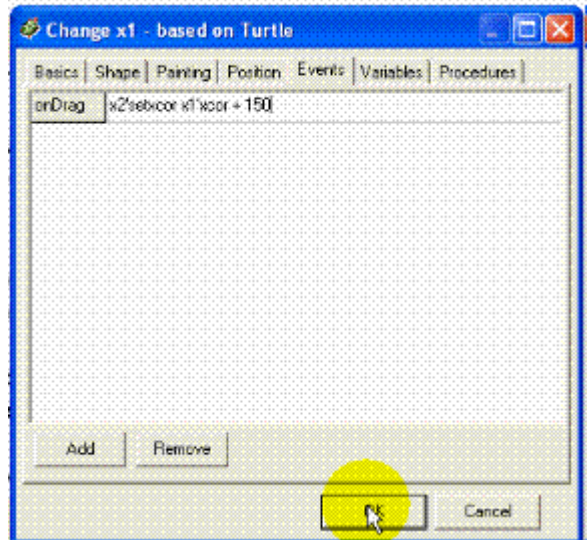


Figure 3. A triggered action associated with “dragging the cat”

In contrast, the specification of the dependency between the positions of the cat and the mouse in the bottom cat-mouse pair in Figure 1 exploits a spreadsheet-style definition feature that is implemented in Imagine-d Logo. For this purpose, the Basics field in the specification for the mouse at the bottom right (here implemented by the turtle x4) is adapted – without any significant change to its layout and format – as displayed in Figure 4. The x and y coordinates of the turtle x4 can be specified in this context not only as explicit values but *by definitions*, so that the x coordinate of turtle x4 is defined to be the x coordinate of turtle x3 incremented by a constant number of pixels. This definition, like a spreadsheet definition, can be directly edited by the model-builder. Figure 1 shows the result of changing the value of the constant increment from 150 to 100 pixels.

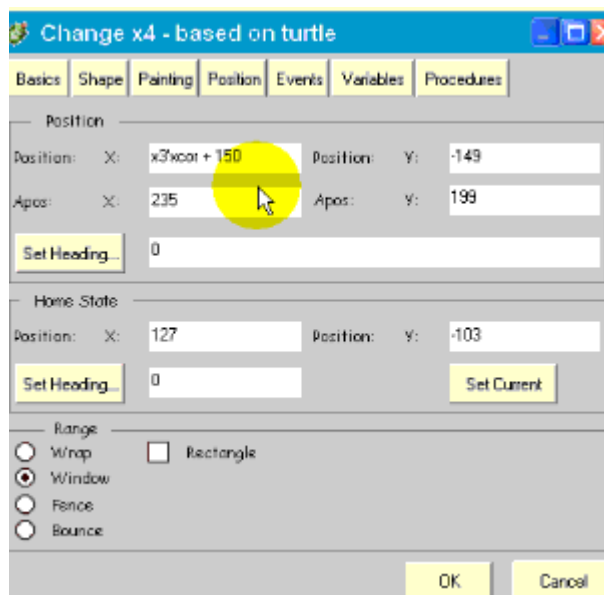


Figure 4. Defining the x position of the mouse via the modified interface

The Imagine-d Logo prototype introduces a conceptually new feature into the programming environment. Where the event-driven paradigm of Imagine Logo requires that relationships are specified by means of triggered actions, the definition of an attribute does not readily admit a simple interpretation as a state-changing action. The implications of this are complex and subtle. It is significant for instance, that defining the x coordinate of turtle x4 with reference to the coordinate of turtle x3 ensures that the value of x4's xcor changes whenever the value of x3's xcor changes, no matter what the nature of the agency that is operating. This contrasts with the discriminating way in which a position update may be associated with a specific event, as in dragging the cat around the screen (cf. Figure 3).

In general, it may be said that definitions afford clearer and more explicit descriptions of relationships though – a superficial analysis suggests – at the expense of some operational simplification. Event-driven sequences of actions can support cyclic patterns of update for instance, whereas the value of an attribute cannot be meaningfully defined in terms of itself. The potential benefits of expressing a dependency through a definition are best appreciated where more complicated patterns of dependency are present. Where a value is defined in terms of many different values, each of which is subject to change through a range of different agencies, the task of analysing the implicit relationships amongst families of attributes can become exceedingly complex, and specifying and maintaining the program structure becomes accordingly more difficult.

Further illustrations

Some additional examples serve to illustrate some of the key issues regarding the introduction of spreadsheet-style definitions to the Imagine Logo environment.

The distinction between the event-driven and definition-based approaches to specifying dependencies is relevant even to simple scenarios, such as maintaining the relationship between the value in a box and the position of a slider (cf. Figure 5). The principal advantage of using a definition is that the model-builder need only inspect the specification of the box object in order to determine why it has its current value. This may not pose a serious problem where the dependency that determines the value in the box relates to the position of a single slider and is kept up to date by a single triggered action. It can become much more problematic when the dependency relations are complex.

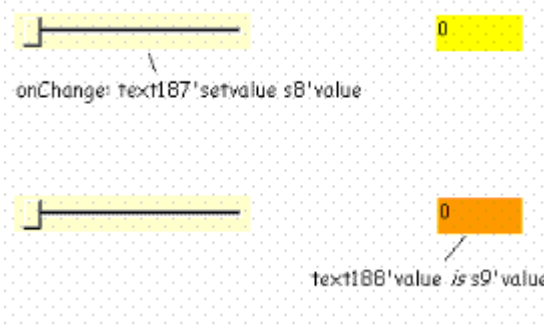


Figure 5. Alternative ways to link the value of a box to a slider

Figure 6 shows a control mechanism drawn from the Shotput microworld (Roe et al, 2005). In this figure, the angle of elevation of the green arrow is determined by the lower slider, and the angle of elevation of the purple arrow relative to the green arrow by the upper slider. The textbox values and the headings of the turtles are then dependent on a combination of the positions of the two sliders. Within the Imagine-d Logo prototype, the heading for the purple arrow can be defined directly by the formula "angleslider' value + attackslider' value". When the dependency is implemented in the traditional way, this formula appears on the right hand side of assignments in two separate triggering events associated with the two sliders. To understand how the heading

of the purple arrow is determined the model-builder has to search for the events that might have an impact upon them. If it becomes necessary to update the relationship, this formula has to be updated in identical ways in two places.

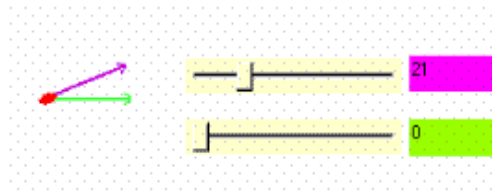


Figure 6. A more complex dependency relationship involving two sliders

Another feature of the Shotput microworld in which complex dependencies are involved is the dynamic graph used to display the x and y coordinates of the shotput trajectory. Because of the wide variety of possible trajectories, it is important to be able to adjust the range of values displayed on the axes, and the granularity of the xy-grid dynamically (see Figure 7). This involves setting up dependencies in which multiple turtles and several textboxes are involved. In this example, dragging the purple point triggers the recalculation of the maximum x and y coordinates that need to be displayed. These positions of the grid lines are dependent on these maximum values, and grid lines are only displayed if they are within the range of the graph. The dependency relationships required to specify the positions and visibility of grid lines are here expressed by definitions.

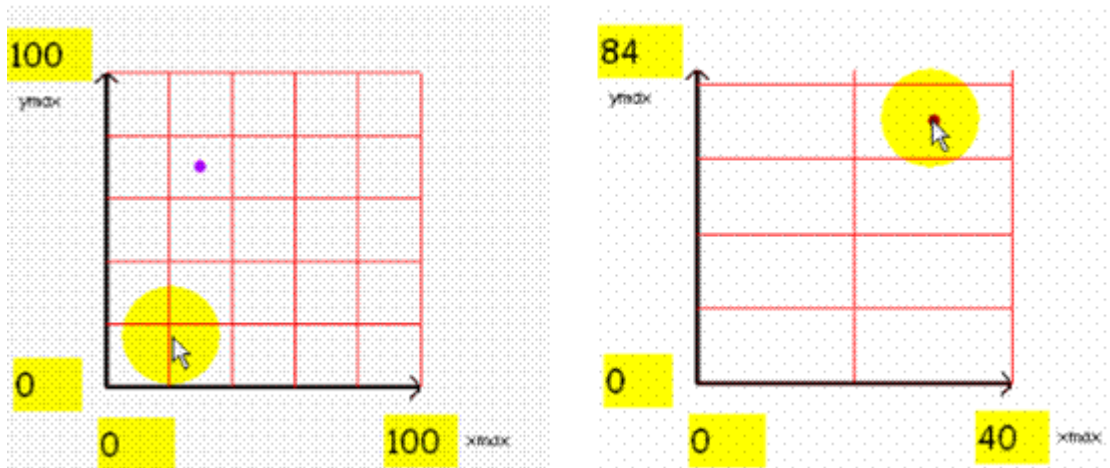


Figure 7. Dynamic graphs implemented using dependencies expressed by definitions

A very rich family of relationships can be programmed by introducing generic dependency maintenance to the Imagine Logo environment. This is evident from the range of applications that can be readily implemented using the Imagine-d Logo prototype. Figure 8 displays another slide from the same presentation from which Figure 1 was drawn. Dependencies framed as definitions have been used in this presentation in several ways:

- To implement a generalised spreadsheet: A simple spreadsheet has been embedded within the displayed slide. In addition to the normal dependencies between values, there is scope to make the presentation of data depend on its content, to exploit slider positions as if they were cell values and to attach names to "floating cells" that do not have to be located within the grid.
- To manage the screen layout: The dimensions, organisation and attributes of windows on the screen can be determined using dependency relationships. The position of bullets in the text can be determined relative to a reference point.
- To animate the presentation: By dragging the dog icon at the foot of the slide from left to right across the screen different right hand panes can be slotted into the slide to illustrate

each of the bullet points in turn. This animation effect is obtained by creating a series of frames that display a different variant of the dog icon according to its horizontal position, and defining the choice and location of the right hand pane in terms of this position.

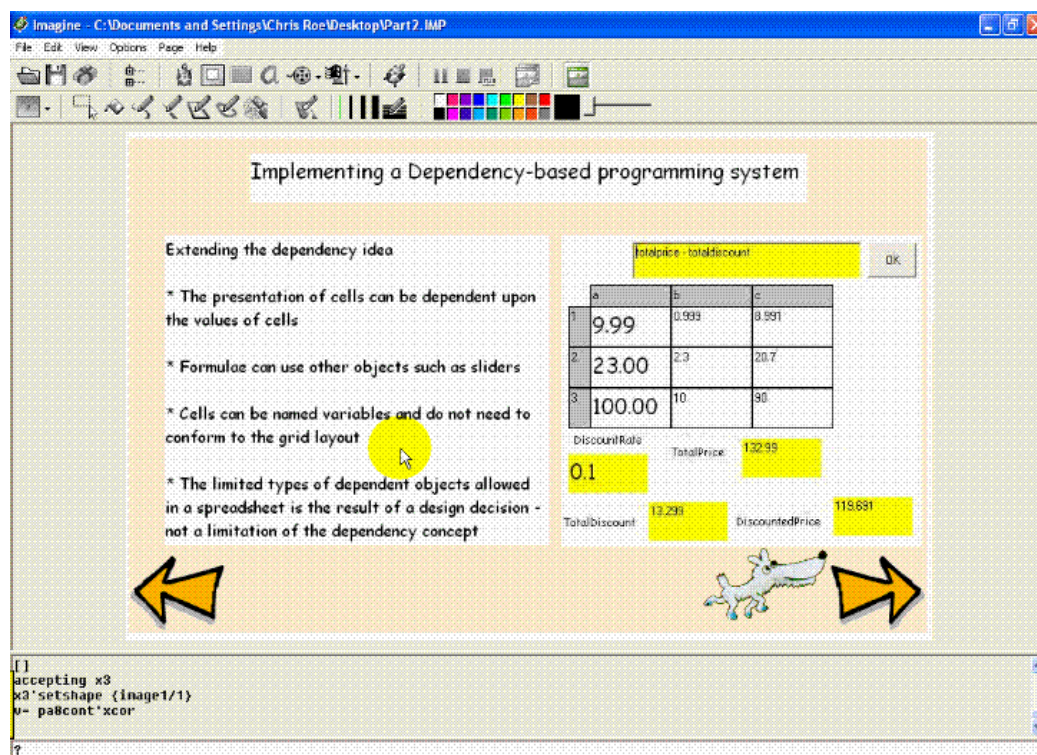


Figure 8. Illustrating dependency in specifying slide content, layout and animation

Discussion

Introducing dependency-by-definition to Imagine Logo has a significant practical impact on model construction. In implementing features such as dynamic graphs, the convenience of being able to formulate dependencies as definitions without needing to explicitly set up the underlying maintenance mechanisms is liberating, and promotes the speed and flexibility of development that is associated with the use of spreadsheets. Extending Imagine Logo in this way seems to be particularly topical when we consider the prominent role that dependency plays in many successful educational products, such as *Cabri Geometry* and *Visual Fractions*. However, in this context, it is worth noting (Kalas, personal communication, 2005) that the designers of Imagine Logo have long been aware of the expressive qualities of dependencies, yet have been uncertain about how these can be effectively and coherently exploited. This is consistent with the fact that understanding the relationship between modelling with dependency and programming has been a central theme of the Empirical Modelling project (EM Web) since its inception more than twenty years ago. Over this period, research on this theme has generated:

- practical tools and models to demonstrate that modelling *purely* with definitions in principle has expressive power and range encompassing what has been illustrated using Imagine-d Logo above (see for instance the presentation graphicspresHarfield2007 (EM Web), in which definitions support geometric modelling and interface management, and implement notations for formulating definitions that are themselves defined within the modelling tool).
- a methodology for modelling with dependency that is associated with a radical reappraisal of thinking about computing (Beynon, 2007; King, 2007).

Relevant issues can be exposed by reviewing "programming with dependency" from each of the three perspectives identified in the introduction: programming practice, pedagogy, and computing science.

Programming practice

It is a conceptual mistake to suppose that adding features to a programming language that contribute to ease-of-development and ease-of-use necessarily thereby enhances its value as a vehicle for constructivist learning. Learning and 'optimisation to achieve a specific programming goal' are activities that engage the mind in quite different ways, and develop in entirely divergent directions. In learning, it is vital to pay as much attention to the environment as possible and to explore every discrepancy between expectation and observation, whereas optimisation involves specialising the task and engineering the context so as to minimise the information we need to know about the environment and eliminate all observation and action that can be made superfluous. The distinction drawn between virtuosity and musicality in performance is one indication that the speed with which we can accomplish specific tasks is not a good measure of how much we have learnt about a domain.

It is also well-recognised that adding features to a programming environment does not necessarily lead to a coherent conceptual enhancement. The challenge to conceptual integrity in Imagine-d Logo can be gauged from comparing the code developed by Tomcsanyi (2003) to implement the dynamic geometry depicted in Figure 2 with the following listing, which specifies the same configuration using the EDEN interpreter – the principal tool for Empirical Modelling:

```

line AB, BC, AC
point A, B, C
AB=[A,B]           // the line joining points A and B
BC=[B,C]
AC=[C,A]
A = {20, 180}      // the point p4
B = {260, 490}    // the point p1
C = {285, 50}     // the point p3
line perpA, perpB, perpC
perpC = perpend(C, AB) // the perpendicular dropped from C on to AB
perpB = perpend(B, AC)
perpA = perpend(A, BC)
point D
D = intersect(perpA, perpB)

```

In both EDEN and the Imagine-d Logo prototype, the programmer can *either* specify the explicit procedural actions that maintain the geometric dependencies in Figure 2, *or* embody them in a set of definitions. But by what criterion should one approach be adopted rather than the other, and is the distinction between these two approaches more than a matter of individual taste?

Pedagogy

In comparing Logo with Boxer, diSessa (1997) stresses the importance of criteria other than abstract expressive power in relation to learning. His concern is as much with the physical form that programs take as with the underlying programming constructs. This is consistent with his observation (diSessa, 2000) that "*make it experiential* is perhaps the single most powerful educational heuristic I know". diSessa also seeks to challenge the assumption that educational software "comes in large units that are so slick and complex as to require many person-years of effort to create them, usually by highly technically competent software engineers".

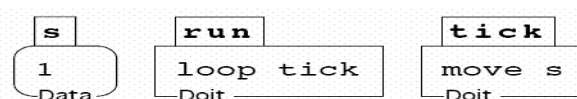


Figure 9. The **tick model** that defines uniform motion (diSessa, 2000)

diSessa's "tick model" of uniform motion, as depicted in Figure 9, reflects his desire to shift the emphasis from programs as complex bodies of code for professional analysis towards visual artefacts that are intelligible to the learner. It also highlights his concern for a close correspondence between the interpretation of the program in the domain and the components of a visual representation. Similar concerns for *experiential* rather than symbolic representations of programs were influential in the design of Imagine Logo (Kalas et al, 2000).

Beynon and Roe (2006) raise a deeper concern about the suitability of programming languages as a means to realise constructionist ideals. The traditional and fundamental understanding of *what a program is* casts the programmer in the role of a specialist with recondite knowledge whose job is to ensure that a program carries out specific functional tasks that conform to explicit user requirements. Under this interpretation, neither the programmer nor the user interacts with the program as an artefact in a way that is well-matched to the agenda of the active learner. The kind of conflation of pupil, teacher, developer roles that constructionist learning ideally demands is much better mapped on to the collaborative moulding of a spreadsheet model to situation by human agents with different levels of domain knowledge and computational literacy discussed by Nardi (1993). Empirical Modelling endorses this idea, privileging the use of sets of definitions to express the latent dependency in situations over the use of conventional procedural constructs and data abstractions to describe processes and behaviours.

Computing Science

The besetting problems of large-scale complex software development are closely connected with the need to develop domain knowledge in an incremental and systematic fashion whilst making computer models in a constructivist spirit. This highlights a strong connection between pedagogical aspirations for software and fundamental concerns in computing science. Empirical Modelling identifies the primitive learning that must underpin software development with a particular way of gaining familiarity and understanding in a domain. This activity has a primary orientation closer to Papert's *bricolage* than conventional programming (cf. Roe (2003), pages 109-116), in that its emphasis is upon modelling concrete situations rather than prescribing abstract behaviours. As discussed in Beynon (2007), this is achieved through making artefacts that are appropriately characterised as *construals* rather than programs. A construal offers interactive experience to the model-maker in which empirically validated patterns of observation, dependency and agency in the domain are faithfully reflected.

The notion of giving computer-support to making construals fits in well with diSessa's conception of *material intelligence* as "an intelligence achieved cooperatively with external materials". The close correspondence between the components of the Boxer program in Figure 9 and their physical counterparts in the domain is well-matched to an analysis of domain observables, dependencies and agents. Generalising such a correspondence to typical computer programs is altogether more problematic. The fundamental historical rationale for the program concept has been that of automating human agency out-of-the-loop. The sharp mathematically defined abstractions that pervade the theory of programming are its legacy. The difficulty of maintaining a credible correspondence between objects as experienced in the domain and objects as they emerge in the design of programs is but one indication of the tension between automated action and reflective thinking. It is clear for instance that the object-oriented programming activity associated with implementing the dynamic geometry of Figure 6 in Tomcsanyi (2003) does not admit the same direct interpretation in geometric terms as the family of EDEN definitions above.

Conclusion

The above discussion highlights the crucial importance in constructionist use of the computer of the experiential aspects of the modeller's interaction. For learning to be effective, there must be a close and – in some sense – directly perceptible relationship between situations, objects and concepts in the domain and the interactive experience of the model-making. In establishing this relationship, what is – or comes to be – hidden and implicit in the interaction is quite as important as what is explicit. Arguably, much learning is associated with acquiring such familiarity with

relationships between significant observables that the procedures for identifying these relationships that at first must be laboriously followed become routine and internalised. But where the polished products of professional software development strive to anticipate what kind of functionality and familiarisation is appropriate and attainable and optimise the user's interaction accordingly, the essence of educational software is that it should expose the raw activities that are associated with identifying functionality and gaining familiarity with procedures.

Dependencies are archetypal examples of relationships that underpin learning. The merits of spreadsheets for educational applications can plausibly be attributed to the fact that – unlike conventional programs – they carry out dependency maintenance activity in a way that is both hidden and intentionally *uninterpreted*. Programming environments such as Imagine-d Logo and EDEN in principle support similar sense-making conventions that allow some “program code” to be explicitly interpreted and some to be as-if-internalised by the model-maker. The problematic issue is that such a policy of blended interpretation and uninterpretation is not consistent with the orthodox abstract conception of computational semantics.

Modern variants of Logo, such as Imagine Logo and Boxer, have taken significant steps towards the concretisation that is essential in supporting a new practice of computing better attuned to the demands of constructionist learning. The sense-making theories needed to enfranchise such practice and to promote its mature use have yet to be identified. As Albirini, citing Gärdenfors and Johansson, remarks: “educational technology still lacks a unifying theory of learning that would serve as a foundation for its use in the classroom” (Albirini, 2007). The motivating thesis behind Empirical Modelling is that such a theory of learning has to be developed in conjunction with a radically new conception of computing (Beynon, 2007).

As diSessa's “tick model” of uniform motion illustrates (see Figure 9), there are contexts in which there is an excellent match between experience of a domain and of its representation in a computer-based artefact. It is evident that the quality of the “tick model” is linked to the fact that the “turtle” metaphor for change is ideally suited to representing the dynamics of physical motion and time. One of the problematic aspects of our current conception of computing is that the professional programmer aspires to a world-view that is equally well-attuned to the very same turtle metaphor. Under this view, software should ideally fit into a world where the processes of change are effected through the continuous action of tamed agents operating under the stable gaze of a benign objective external observer. In this world, constructing software is uncovering the theory that governs the mind as machine and all change as ultimately explicable with reference to eternal absolute physical laws.

Papert's vision for constructionist learning is not well-served by this limited understanding of computation. Such an objectified conception of change needs to make room for an altogether wilder perception of the perspectives and vectors that are associated with learning: their subjectivity, potential incoherence, and the absence of well-rehearsed and practiced trajectories. The primary emphasis in computing for this purpose is not on processes and behaviours, but on situations in which human interpretation operates in flux, uncertainty and potential confusion, and on construals rather than programs.

The above reflections suggest two different ways in which the Imagine-d Logo prototype might contribute to future research. With a sufficiently robust and comprehensive implementation, the dependency front-end could become a standard feature of the Imagine Logo environment, to be used in a discretionary manner as users see fit. The examples in this paper have illustrated the potential for using such a front-end in several different ways: to link the contents of a screen display to internal values, to set up dependencies between turtle positions, or to define simple dynamic geometry configurations. There is scope for school-based empirical testing to explore the merits of such a mixed-paradigm approach. A more radical research direction would involve trying to blend the fine qualities of tools such as Imagine Logo and Boxer where accessibility is concerned with existing Empirical Modelling tools such as the EDEN interpreter (as discussed in King, 2007). The extensive Empirical Modelling project archive (EM web) already demonstrates proof-of-concept for dependency-by-definition in a wide range of applications, but as yet has had

little exposure where users without specialist computer science knowledge are concerned. Given the Jamesian philosophical stance of Empirical Modelling, a better metaphor for mental activity in that context – rather than a turtle in motion – might be that proposed by Naur (2001): *an octopus jumping in a pile of rags*.

References

- Albirini, A. (2007) *The Crisis in Educational Technology, and the Prospect of Reinventing Education*, Educational Technology & Society. 10(1), 227-236
- Baker, J.E. & Sugden, S.J. (2003) *Spreadsheets in Education - the First 25 Years*. e-Journal: Spreadsheets in Education 1(1): 18-43
- Beynon, M. (2007). *Computing technology for learning - in need of a radical new conception*. Educational Technology & Society. 10(1), 94-106
- Beynon, W.M., Roe, C.P. (2006) *Enriching Computer Support for Constructionism*. In Alkhalifa, E. (ed.) *Cognitively Informed Systems*, 209-233
- Cabri geometry: <http://www.cabri.com/v2/pages/en/index.php> (27th March 2007)
- diSessa, A.A. (1997) *Twenty reasons why you should use Boxer (instead of Logo)* In M. Turcsányi-Szabó (Ed.), *Learning & Exploring with Logo: Proceedings of the Sixth European Logo Conference*. Budapest Hungary, 7-27
- diSessa, A.A. (2000) *Changing Minds: Computers, Learning and Literacy*. Boston: MIT Press
- EM Web: <http://www.dcs.warwick.ac.uk/modelling> (28th March 2007)
- Kalas, I. & Blaho, A. (2000) *Imagine...New generation of Logo: Programmable pictures*. In Proceedings of WCC2000, 427-430
- King, K. (2007). *Uncovering Empirical Modelling*, MSc, Computer Science, Warwick University
- Logo Foundation. *What is Logo?* <http://el.media.mit.edu/logo-foundation/logo/index.html> (25th March 2007)
- Nardi, B. A. (1993). *A small matter of programming*. MIT Press
- Naur, P. (2001) *Anti-philosophical Dictionary*. naur.com publishing
- Perry, M. (2001) *Automate Dependency Tracking*, Parts 1, 2 & 3, JavaWorld.com, Aug-Oct 2001, <http://www.javaworld.com/javaworld/jw-08-2001/jw-0817-automatic.html> (26th March 2007)
- Roe, C.P. (2003). *Computers for Learning: An Empirical Modelling perspective*. PhD thesis, Computer Science, The University of Warwick
- Roe, C.P., Pratt, D., Jones, I. (2005) *Putting the learning back into e-learning*. In Proceedings of CERME4, Sant Felix de Guixols, Spain
- Sendov B., Dicheva D. (1988). *Mathematics Laboratory in Logo Style*, In Fr. Lovis & E. Tagg (eds.), *Computers in Education*, IFIP, ECCE'88, North Holland, 213-217
- Tomcsanyi, P. (2003) *Implementing object dependencies in Imagine Logo*, In Proceedings of EuroLogo 2003, 27-30 August, Porto, Portugal
- Visual Fractions: <http://www.logo.com/cat/view/logotron-visual-fractions.html> (27th March 2007)