Mathematics with component-oriented exploratory software

Chronis Kynigos, Manolis Koutlis, Thanasis Hadzilacos

Abstract

In this paper we discuss a component-oriented architecture which we are employing to develop programmable exploratory software for mathematics. We argue that the architecture can be used to provide synergy between end-user programming and efficient behavior of components, i.e. computational objects of a wide range of technical complexity and functionalities. We give examples of components with mathematics in their behavior and components which in themselves embody mathematical relations. Through both formal language and visual means, users can link them to form creative configurations with interesting functionalities and use the resulting environments for exploratory activity. We conclude that this architecture enables a more efficient collaboration between technical and educational expertise in developing exploratory software.

Keywords: component software; exploratory learning environments; variation tool

Introduction

As new features and functionalities are continually appearing in educational software, programmability and extensibility are loosing their importance if only because they are becoming “options” among many. Yet these are both important for mathematical expression, exploration and meaning construction. We argue that instead of being supplanted, these notions could provide new ways of taking advantage of new technologies in exploratory software. To do this we make a case for programmable component-oriented software and use descriptions of our relevant current work to elaborate on the ideas. We begin by making a case for programmability, then we do the same for component architecture and then for their merging to construct mathematical microworlds.

The idea of computers as a powerful means for mathematical expression via formalization emerged at a time when text and numbers were considered the main (if not the only) representational method to do something with a computer and function-derived graphics were the only alternative representation provided as computer feedback. These were also the days of keen interest of the mathematics education community (at least in the U.S.) in perceiving “problem-solving” as a generalized ability for doing mathematics irrespective of the task, the situation and the context (Noss and Hoyles, 1996). The emergence of a multitude of alternatives for educational computer use for mathematics and other subjects and the fading out of perceiving “mathematical ability” as a useful means to think about and encourage mathematical learning played a major role in reducing the interest in formal expression with a computer (for a related discussion see also Kynigos, 1995). By many, it came to be considered as both technically and educationally out of date especially in areas of education outside mathematics: related rhetoric supports that formalization and the necessary abstraction that goes with it, can be bypassed now that we have tools allowing us to iconically represent structures and control complex program behaviors with direct manipulation. Since understanding is situated, let us perceive it as happening only within situations; the abstraction required for formalization is too high to be used for meaningful learning.

For mathematics education, we see a prominent question here, which is both pedagogical and epistemological in nature: is formalization to be perceived as (just) one of the ways to represent mathematical ideas - the hardest one in fact - or as an essentially mathematical activity in

itself? We certainly appreciate the value of software now available with modern interface technology, focused on combining alternative to formal language representations such as real physical motion with velocity-position graphs, or real-life simulations with graphical representation (Noble and Numirovsky, 1995, Roschelle and Kaput, 1996). But we also feel the need to provide pupils with the opportunity for expressing intuitive or situated ideas formally (Kynigos et. al., 1993). As Noss puts it, we could on the one hand bypass formalization or on the other, we could exploit it for mathematical expression (Noss, in press). Over a period of more than ten years of research, studies have shown that formalization does not necessarily mean abstraction beyond what pupils are capable of. On the contrary, instead of perceiving abstract thinking as a prerequisite for formal expression, we have important evidence of pupils mathematizing by means of what Hoyles and Noss termed "situated abstractions" (1992, 1996) while incorporating formal language in representing ideas.

We suggest that the enrichment of representation, functionality and feedback achieved with new technology does not in itself make formalization less important for mathematical expression. On the contrary, new technology could both empower this kind of representing mathematics and enable more focus on the pedagogical design of environments for constructing mathematical meanings. There are eloquent examples of "programmable" software, built around the idea of formalization, that exploit innovative technologies to provide learners with powerful expressive media and metaphors. Amongst them, Boxer incorporates "naive realism" as a metaphor for a computational expressive medium (diSessa and Abelson, 1986). The idea of domain specific extensions to a programming language for synergy amongst conventionally fragmented functionalities is suggested through Schemepaint (Eisenberg, 1995). Starlogo, uses parallel processing metaphors to provide a vehicle for decentralized thinking (Resnick, 1995); and several others, like Microworlds Project Builder, Function Machines (the list is by no means complete).

It is in this context that we would like to build a rationale for a software environment for exploratory mathematics based on the notion of "components", that provides formal ways of expression while maintaining a high level of functional efficiency and potentially exploits state-of-the-art software technologies. Components are autonomous, reusable computational objects playing the role of primitive (albeit not quite) building-blocks whose behavior can be "programmed" at end-user level. They are tangible software entities that can bear technically or functionally, simple or complex behavior. In either case, their special characteristic is that they are designed so that they may be combined with other components in configurations defined by end-users and behave as composite structures. For mathematics, the degree of freedom in defining and designing a component allows us to rethink of the possible functionalities of computational objects so that they can be reused in a much wider variety of situations.

Our initial approach in adopting a component-oriented way to develop exploratory software, involved designing components for geography education (Koutlis and Hadzilacos, 1996). However, in this paper we argue that the new technical capabilities available for constructing components present opportunities to infuse mathematics in their behavior and to reconsider the definition and functionality of some mathematical objects themselves. While being in an ongoing process of research and development, we would like to suggest that the concept of programmable components both as a development architecture and as a means for creating mathematical environments is useful and promising. To do that, we will use examples of components we are currently designing.

So, we begin by outlining why we think a component approach may be used to develop exploratory software. We then consider a number of somewhat classic components for mathematics and how they could function in synergy providing new functionalities. In the next section we describe a specific mathematical component, a variation tool which is put in use to control variability within or between components. Finally, we give an example of a geographical microworld tuned (to use Noss' expression, in press) to invoke mathematical

thinking, in an attempt to show how mathematical behaviors can be embedded in a wide variety of microworlds.

Component architecture for exploratory software

Programmability is important in exploratory software, not only as a means for learners to formalize mathematical ideas as we argued in the previous section, but also as a powerful means (in the sense that it is a language rather than a set of options, Eisenberg, 1995) for the end-user community (researchers, teachers, pupils) to construct and reconstruct objects and functions (diSessa and Abelson, 1986, Harel and Papert, 1990, Harvey, 1993). The approach taken towards that end in environments like Logo and Boxer, as well as in commercial end-user programming systems like Hypercard™ and Toolbook™, is to provide an interactive programming language as their core tool, possibly augmented with domain-specific primitives. But although powerful in the sense of the generality of scope and the range of applications that can be realized, users are confined to either start constructing from scratch in terms of primitive entities and operations, or rely on other users' constructions within the *same* environment.

Building technically complex objects and functionalities is a specialized task that grows exceedingly harder as the target requirements rise. End-users are thus faced with a dilemma: they either need to develop technical knowledge and devote considerable amounts of distracting time only to achieve mediocre results, or hire a professional programmer who will most probably will work with tools of his/her own choice (different from the end-users') and come up with efficient but otherwise monolithic modules, not reusable out of their specific design context. In both cases, it is very unlikely to achieve something beyond fragmented and single-use software, resulting in a poor and ineffective realization of otherwise eloquent ideas (Kaput and Roschelle, 1996).

What is missing is a way to provide end-users with a repertoire of high-level construction building blocks, integrated as available objects within a programming language, allowing educators to focus more on the pedagogical issues of a microworld's design and learners to manipulate and receive feedback from a much wider variety of representations and behaviors. In other words, teachers and children should be given the ability to build and think in terms of objects that are close to their domain of interest, susceptible to manipulation, inspection and programming of their behavior. A central argument of this paper is that software built with a component-oriented approach can contribute towards these goals.

With components, microworld developers can start thinking about the design and construction of their own artifacts in terms of high-level entities that encompass both a domain's modeling and the accompanying functionalities. Confronted with the task of building or restructuring a microworld, they can choose from a pool of prefabricated components and define any series of actions or inter-component interactions that achieve the desired functionality, either symbolically through a language or by direct manipulation methods. In that case, they deal with "programmable behaviors" besides data-models and algorithms, while microworld construction incorporates the notion of assembling a "kit" in that traditionally thought of as programming.

For example, in order to create an exploratory activity where pupils could experiment with the notions of time differences around the globe and the relevance with position and geographic scale, all that would be needed in a component-oriented environment, is the set of suitable components and a way to "connect" them: a Globe, a Clock, a Map and a Turtle able to travel around the globe, accompanied by a navigation component (which could be Logo for example augmented with spatio-geographical commands). Once connected, the components work in synergy: the Clock displays the Turtle's local-time, the Globe is shaded accordingly, the Map zooms around the Turtle's position, and so on (we elaborate on a similar example later on).

What is important is that components should be designed as autonomous software entities with their own attributes, user-interface, structure and functionality, capable for standalone use if this is what is required, but with the additional capability to expose pieces of their internal structure, either to other components (through connection "plugs"), or to users (through language primitives), thus allowing for introspection and sharing of data and behavior. This would result to tight synergy and interoperability among components, large scale reusability of the prepackaged functionalities (and thus saving of resources), freedom for synthesis or decomposition of constructions at component level by end users, high degree of microworlds' programmability. Further, in an anticipated component-oriented world where design standards would be established and widely adopted (Roschelle and Spoher, 1997), components developed by different teams would be readily affordable to everyone for incorporation in personal configurations, thus achieving the ubiquity of excellent pieces of software which otherwise remain in the isolation of their custom design and way of use (one could expect for example that a Graphing component would be developed once by a specialized team, and then shared by users).

This approach to software development is not without cost: component design is much more difficult than that of designing a standalone applications. Since a component's possible uses and connectivity arrangements are practically unpredictable, its functionality should be specified as context-free as possible, its behavior should encapsulate an abstraction of its anticipated instances, and its communication interface (i.e. which functionalities or data are exported, imported or shared with other components) should be clearly defined. In the geographic scenario for example, questions and trade-offs that have to be addressed in the design process include: should the Turtle's navigation algorithm be encapsulated in the Map component, in the Globe, in the Logo, in the Turtle itself, or should it breakdown to all of them? (bearing in mind that these components will not necessarily always participate all together in microworlds). Should the same Clock be allowed to be connected (by end-users) to more than one Turtles at a time, or could this lead to inconsistencies? Should the Turtle's path attributes like color and width be manipulated from user-interface controls implemented by the Turtle component itself, or should this functionality constitute an individual component (a Palette) that would be plugged to Turtles but also to any other component that needs this kind of user input? Do Map's functionalities refer to the right conceptual level for all of the potential users and anticipated contexts, or should they be redefined to a more generic level?

In any case, end-users should be provided with components,

- whose behavior stands at the level of "atoms" -in the sense of "no further decomposability"- for their domain of concern (for example "map" is a component that needs not be decomposed to its constituent data-structures and algorithms for a geography teacher),
- that are flexible and generic enough to allow for a variety of uses
- that allow for introspection of their sharable attributes and functionality at a level that is both meaningful to end-users and consistent across all other components.

Components for Mathematics

As mentioned earlier, we began considering the potential of component architecture for mathematical software environments based on prior work accomplished for the domain of geography education (Hadzilacos and Koutlis, 1993). Although it is relatively easy to identify tangible geography components like maps, compasses, globes and clocks, it is somehow elusive to conceive of mathematical components that model more abstract concepts. One way to start is to think in terms of core functionalities of exploratory mathematics software available today and consider an environment where these would be components for mathematics. So, we could have:

- A Calculator component with programmable buttons.
- A Spreadsheet component able to import, export and manipulate data in table format.
- A Database component, providing the ability to organize, structure and manipulate data.
- A Charting component that would import and plot either arbitrary data or functions (Roschelle and Kaput, 1996).
- A Graphics component providing turtle graphics, dynamic manipulation of Euclidean graphical constructions (a Cabri-like functionality), a graphical manipulation and depiction of sets of objects imported from other components (Tabletop-like functionality, Hanckock, 1995) and tools for interactive drawing.
- A Logo-like parser component providing a language for functional expression, and a Prolog-like parser component allowing for experimentation with deduction and logical inference.
- A Turtle component, playing the role of an agent controllable from command-providing components and operating on graphics or simulation components.
- A Sets component that would allow graphical manipulation and depictions of sets of objects imported from other components (Tabletop-like functionality).
- A Symbolic algebra component with Mathematica-like functionality (Wolfram Research Inc., 1996).

In addition to components directly relating to mathematics content, there's a whole-lot of useful components like for example Clocks and Calendars, Radios and TVs, Maps and Globes, as well as a number of others modeling objects from various science curriculum domains that could be used in a mathematical context. The point is that since this architecture allows tight interoperability amongst components, they would be able to synergistically operate as user-defined entities incorporating clusters of components. Each cluster would behave as a coherent whole, combining the components' functionalities in configurations that otherwise may have not been possible.

As a simple example, in order to develop a Logo-based software for mathematics which we describe in the following section we needed the functionality of turtle geometry. To do this in our environment[1] one has to simply plug a "Language" component[2] to the "Turtle" component, and the latter to a "Canvas" component (we could of course have more than one of any of these if required). This is done by using the component's plugs as shown in figure 1[3].

In the same way, and with the right set of components we could set-up environments that incorporate the functionality found in existing successful software like for instance Tabletop (Hanckock, 1995) and FunctionProbe (Confrey and Maloney, 1996). We could for example have a Logo-database environment where the formal and/or the graphical results of procedure executions are fed both to a "Database" and "Canvas" components and be manipulated in different but synchronized ways. Or, as we describe later on, the "Canvas" component of the previous example could be substituted by a geographical "Map" component, and provide the functionality of formally describing turtle trips in representations of physical environments with the respective restrictions and properties.

---

[1] We refer to "Avakeeo", a software platform designed to support the construction of exploratory software with components (http://www.cti.gr/RD3/EduTech/avakeeo.html).

[2] A porting of Berkeley-Logo.

[3] Only plugs of the same color and matching shape can be plugged.

Moreover (and most importantly), as a component is incorporated in a Microworld it "automatically" integrates with the underlying language by adding component-specific primitives which control its functionality and can be utilized by end-users in their programs. In that way, almost any aspect of a component's hardwired behavior can be unleashed according to the respective pedagogical design and be made available to end-users through a language-based, symbolic interface.

Mathematical Components: A variation tool

As we began to argue in the previous section, the notion of a mathematical component cannot be similar to that of, say, a geography component; mathematics is the field of relations and processes, so it cannot be described as a set of interoperating objects. The components for mathematics which we already described are objects with and within which relations can be expressed. In this section we present another approach, complementary to the one above, that of components, or mathematical objects, which in themselves play the role of relations amongst and/or within components.

As an example, we discuss a component we call a "variation tool", which we've designed to represent a metaphor for manipulating variation within or between components. We believe this kind of functionality is deeply rooted in important mathematical ideas such as change (Rochelle and Kaput, 1996, diSessa, 1995) and that the level of abstraction in defining it so that it captures the depth and breadth of mathematical change is by no means trivial. We describe the tool's functionality as we have developed it so far, in order to also make the point that the component architecture allows us to rethink such functionalities and the mathematics within them. Our approach is pedagogical; in order to show some of the potential of its use for learners and microworld designers we set its description against the well researched problems pupils have in understanding variation in the main areas of mathematics. We also provide a brief background of the main functionalities of the software available so far in each mathematical domain. Subsequently, after describing the way in which the tool works in a Logo setting, we give examples of its potential use in Logo-based microworlds. Finally, in the next section, we refer to the tool in a setting linking mathematics with geography.

Variation is at the heart of very important areas of mathematics and many notorious problems pupils have in bringing meaning to mathematical concepts are related to not perceiving mathematical objects as instances of a class of objects with variants and invariants (Hoyles, 1991). Pupils often do not expect or perceive the mathematical in variation, and have difficulty with its conventional mathematical representations and with being able to abstract from a specific instance of something changing (or a descrete number of instances) to the process, the rule or the system with which change can be explained. And yet, identifying invariants across experiences coming from actions on our physical and social environment is a key feature in the way we learn (Sinclair, 1987, Vergnaud, 1982, Lave, 1988) and the basis of many intuitive mathematical understandings. Curricula approaches to mathematics incorporating variability are traditionally insensitive to building on pupils' intuitions. The variation tool is designed to fall in the category of software tools allowing pupils to use their intuitions to gain insight into the mathematics of variation.

In geometry curricula, the starting point is often a class of mathematical objects -the definition of "a figure" representing the class by means of its invariant properties- represented either by algebraic symbols or by an iconic instance of the class. Pupils are given little chance to notice invariance among different instances of a class of figures. Research shows that very few pupils assign properties to iconic representations at all (Fuys et all 1985) and that most pupils confuse a representation of a class of objects with the object itself. Geometry software like Cabri and Scetchpad (Jackiw, 1990) has provided pupils with the means to manipulate the iconic representation of a figure causing a continuous change effect restricted by the properties built

in the initial construction. Language based software like Logo and Geomland (Sendova and Sendov 1993) allow formal definitions of instances of figures and of classes of figures (e.g. by means of a parametric procedure) and their use for the creation of instances of their graphical representation.

In algebra, variability underlies many areas taught in schools like the notion of variable, functions, differentiation. Regarding the former, pupils have difficulties with understanding the notion of a name assigned to a variable and the use of letter notation for naming variables (Kucheman, 1981). They find it difficult to accept unclosed algebraic expressions (e.g. 3+5a) and to understand that a systematic relationship exists between two variable dependent expressions (Booth, 1984, Sutherland, 1989). Logo and spreadsheets have provided pupils with the means to formally represent meaning by applying informal methods with formal language (Hoyles et. al., 1991). Pupils express mathematical processes incorporating variables and then create instances of the result of these by invoking them providing a fixed value for the variable each time.

With respect to functions, Dubinsky has identified three landmarks of pupil abstraction: a) function as action, i.e. a set of isolated calculations, b) function as process, i.e. a sequence of calculations and c) function as object, i.e. encapsulating these processes into objects and noticing their behavior and relations between them. Cuoco relates these understandings to notions of function with respect to three different ways in which it is used, i.e. continuous change, algorithm and mapping. Software designed for exploring functions encourages concurrent representations such as symbolic (algebraic), coordinate graphical, tabular, iconic. It is also based on the idea of function machines, either explicitly or implicitly. These are representations of the process, the input and the output of a function.

Function Machines for example (Feurzeig and Richards, 1996), provides iconic representations of function machines and links amongst them. It is designed to encourage pupils to interiorise actions into processes (Cuoco, 1993). Logo allows the construction of function machine procedures providing rich experience with the notion of function as process. Function Probe (Confrey and Maloney, 1996) combines tabular, graphical and symbolic representations and seems useful regarding the interplay between function as action and function as process. ISETL is a symbolic language for considering functions as objects, with which constructions and investigations with higher-order functions can be made with mathematical symbols and syntax. Mathworlds (Kaput and Rochelle, 1996) enables the construction of experimental situations relating graphical to physical representations of motion. It emphasizes the creation of meaning in graphically representing physical phenomena.

Another way of thinking about variability is the degree of emphasis on phenomenology. If we take a phenomenological approach, pupils form naive understandings (DiSessa, 1987) on how parameters or aspects of a phenomenon may vary based on their experiences. They have difficulty in discriminating variants from invariants and identifying the effect of change on the phenomenon. If we take a mathematical approach, pupils have difficulty in conceptualizing systematic relations between variants and invariants and in understanding covariation. They also find it hard to abstract relations from a set of specific instances of related values to a process or a rule describing the relation and as a final step to perceiving the relations themselves as objects or entities.

Our attempt is to define a variation tool which would be functional in all the above content areas and at the same time provide a common metaphor for handling variation. Our intention is that manipulation of the tool, would provide one or more representations of how an entity behaves both when a parameter changes, but most importantly, *as* it changes. The variation tool can be connected to any other component which is designed to incorporate some systematic variability which, as educationalists, we want users to use, explore and discriminate. When connected, the tool provides the user with a direct manipulation metaphor for sequentially

changing the numerical value of any varying parameter in a component (or a combination of components) and simultaneously observing the behavior of the parts of the component(s) being influenced and the invariability of those that are not. Currently in the form of a slider (Kynigos et. al., 1994), the variation component allows the user to decide on the numerical range of values and the step from one value to the next; it provides a feeling of manipulating or controlling the evolution of a phenomenon or an object while changing one or more parameters in a tangible understandable linear and interactive way.

We describe an example of the tool's breadth of functionalities in the specific setting of building Logo-like microworlds (Kynigos et. al., 1997). The tool can be added to the Logo-turtle environment of the previous section by simply plugging it to the "Language" and "Canvas" components as shown in figure 1. In Turtle Geometry, it behaves like this: clicking the mouse on any part of the trace of a graphical representation of the execution of a parametric procedure with a specific value "energizes" a variation component which provides a slider for each of the parameters of the procedure (Kynigos, et. al. 1994). Each slider has an editable range and step and a tracking feature. Dragging the button on the slider "continually" erases and re-executes the procedure with the value corresponding to each position of the button. The user observes the graphical representation of the parametric object as the parameter changes sequentially. For example, applied to a procedure for a square (figure 1) with a variable for its sides would have the effect of the square growing and shrinking. Variation is thus conveyed not as a set of instances of a varying object but as an evolution of that object.

The effect is in fact impossible to represent by a static figure on paper. In figure 1, moving the slider from the value of 30 to the value of 50 causes the effect of a square's sides continually growing from 30 to 50, and the highlighted numerical input to roll sequentially with a step of 1.

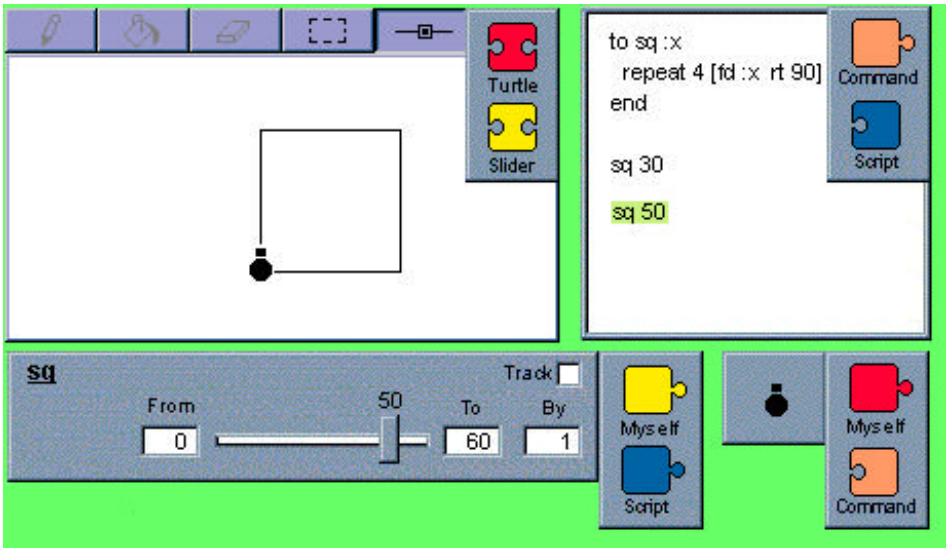

Figure 1. A Component Logo environment with a Variation Component

Such change could of course be implemented automatically by executing a super-procedure eternally calling the procedure with new values. The variation tool provides the user with direct, kinesthetic control of the process.

Already, there is a range of ideas for mathematical microworlds which, because of this direct control, tightly link enactive, graphical and symbolic representations. One group of ideas is procedures creating traditional geometrical representations which can be used or changed and studied by pupils. An example is an "angle" representation procedure built upon variations of the procedure shown in figure 2:

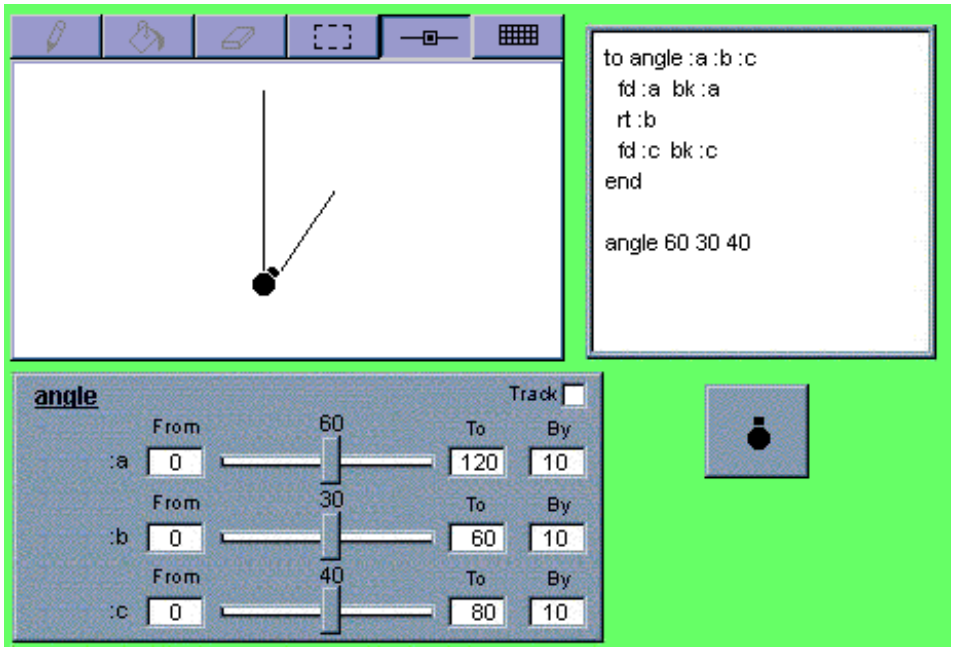

Figure 2. An Angle Microworld with the Variation Component

This code simply causes the turtle to move forward and back the same variable number of steps (variable :a in the figure), turn a variable number of degrees to the right (variable :b) and then go forward and back a third variable number of steps (variable :c). In this way, the turtle "walks" along the classical representation of an angle. Manipulation of the second slider provides the effect of the changing angle between the two segments - the segment on the right seems to be rotating. The pupils could thus construct a meaning for angle based on dynamically manipulating its representation, the static version of which they so often bump into in school mathematics. In a recent case study (Kynigos, 1977) two groups of 12 year olds, were involved in a variety of activities with the microworld: moving the sliders, executing the procedure with specific values or changing constants and/or variables within the procedure itself. Pupils thus have a mathematical construct which they might use either as a tool by observing the effect of moving the slider or as an object by changing and studying it. For example, a pair of pupils in the study changed the procedure to construct a similar one with jagged sides, called it "an eagle" and simulated the flapping of its wings with the dragging of the mouse. Constructing procedures like this as a microworld design strategy is elaborated in Hoyles and Noss (1987).

Another microworld example is geometrical figures based on their properties, such as the following parallelogram procedure (Hoyles et. al. 1991):

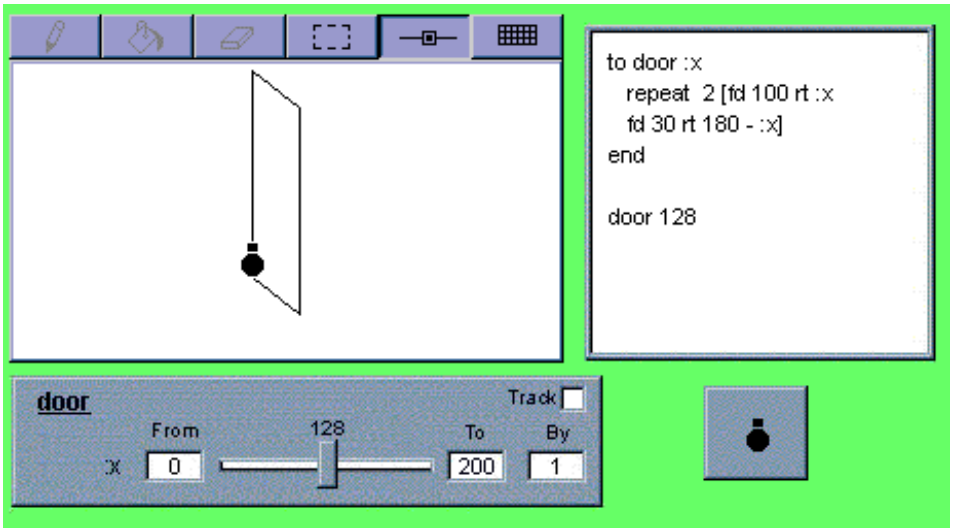

Figure 3. A Parabola microworld with the Variation Component

The pupils can cause simultaneous and “continuous” change in the figure and the numerical value of the input, get a feel of what changes and what stays the same and change the definition of the procedure to try out other hypotheses. For instance, the procedure could be given to them with a different variable for each turn so that they could experiment with the two Sliders to find pairs of values for which the figure becomes a parallelogram and make their own generalizations.

In a rectangle/hyperbola microworld, Cartesian coordinates and function graphs can stem out of enactive graphical representation of the evolution of a rectangle with a constant area:

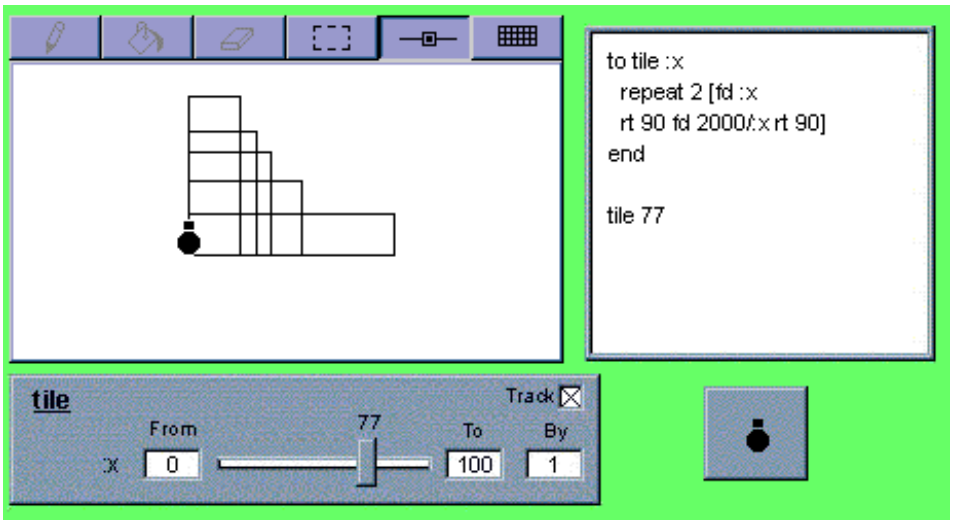

Figure 4. A Rectangle/Hyperbola Microworld with the Variation Component

The top right vertex of the figure represents the values of f(x)=a/x in a coordinate system with the bottom left vertex as origin. Thus within the same microworld, pupils could begin from walking around a rectangle path and end in constructing some meaning for function and coordinates.

In a functions and graphs microworld, a procedure which draws a dot on a point located by its x and f(x) coordinates would make the slider move the dot on the trace of the graphical representation of the function (a “trace” facility on the slider would draw the curve). So, any change in the f(x) expression inside the procedure would create a different path for the dot. Furthermore, a new variable input could be introduced representing a parameter for the

function (e.g. f(x) = ax) and the slider be made to manipulate the parameter making the changes in the curve observable. In fact, one could get as many sliders to appear as there are variable inputs and even have two dimensional sliders affecting two parameters "simultaneously" (a 2-d slider was in fact being constructed at the time of writing). In this way, variation tools could reflect vectors.

The examples we gave relate symbolic representations to "pure" mathematical iconic representations. The component environment makes it possible to embed the graphical representations into representations of real life situations. The "eagle" procedure is an example. Another is the parallelogram investigation which could be in the form of finding the range in the slider so that it opens and closes the front door of a house -the door representing the figure. The house and its surroundings could be made by freehand drawing with a palette component (see Eisenberg, 1995), by a Logo construction, by a scanned picture or by some multimedia gadget making the door creak etc.

Variation is not the only mathematical meaning which could be embodied by a component. This particular component provides a kinesthetic feeling of behaviors during change and of the rate of change. In this sense, it can be described as a differentiation tool. What changes can be represented graphically or numerically and it can be used in Geometry (like in the angle microworld) or in Algebra (like in the hyperbola and the function microworlds). Another mathematical component could be, for example, an integration component, providing metaphors for identifying, covering and measuring areas.

The interoperability between components made available with this architecture relaxes the technical restrictions in thinking about synergy between functionalities which are conventionally attributed to different software. As we already mentioned, we could, consider composite Logo-like/table/graph/calculator tools for mathematical expression. In such an environment, the graphical or symbolic instances created by moving the slider in the previous examples could be "sent" to a "table" component - rather like points on a graph can be "sent" to a table in Function Probe (Confrey and Maloney, 1996). In this case, however, we might want the table to "know" which procedure the value or figure in a cell came from.

The mathematical in component behavior: The day & night microworld

In the two previous sections, we have discussed two kinds of components directly related to mathematics. The component approach enables us to think of new kinds of environments where mathematics may be embedded in a variety of situations. In this way, we have new potential for the development of tools which foster interdisciplinary approaches and at the same time allow for focused investigation. The new kind of technical freedom provided in conceptualizing the tools, their functionality and applicability poses new problems on what is (or could be) mathematical in their structure and their use.

In this section we consider two more kinds of components, those designed to simulate physical, geographical (or other) phenomena and their respective behaviors and those containing fields of information represented in various ways. All of the former "behave" in a certain way where some things can change and others cannot. With the proper design, the means with which users can make changes during their explorations could invoke meanings for systematic or mathematical variation in the sense analyzed in the previous section. The latter kind of components lend themselves to searching for information (finding places on a map, information in an electronic book, etc.) How can these be tuned to invoke mathematical as well as other meaning through their use? In our following example we describe a geographical environment we are developing in this context.

The pedagogical goal is for children to explore representations of the notion of day and night, time-variations around the globe, the shape of the earth and its correspondence with map

projections and the system of global positioning. In addition to the Language, Variation and Turtle components pupils are now provided with a Globe, WorldMap and Clock components. These are configured in a way that the WorldMap's day-night areas are also reflected to the Globe's view (achieved by just synchronizing the GMT-times of the two components), the Turtle's positioning is reflected on both the WorldMap and the Globe, and its corresponding local-time on the Clock's reading. In that way, any change in a component's state is automatically reflected on any other dependent components accordingly.

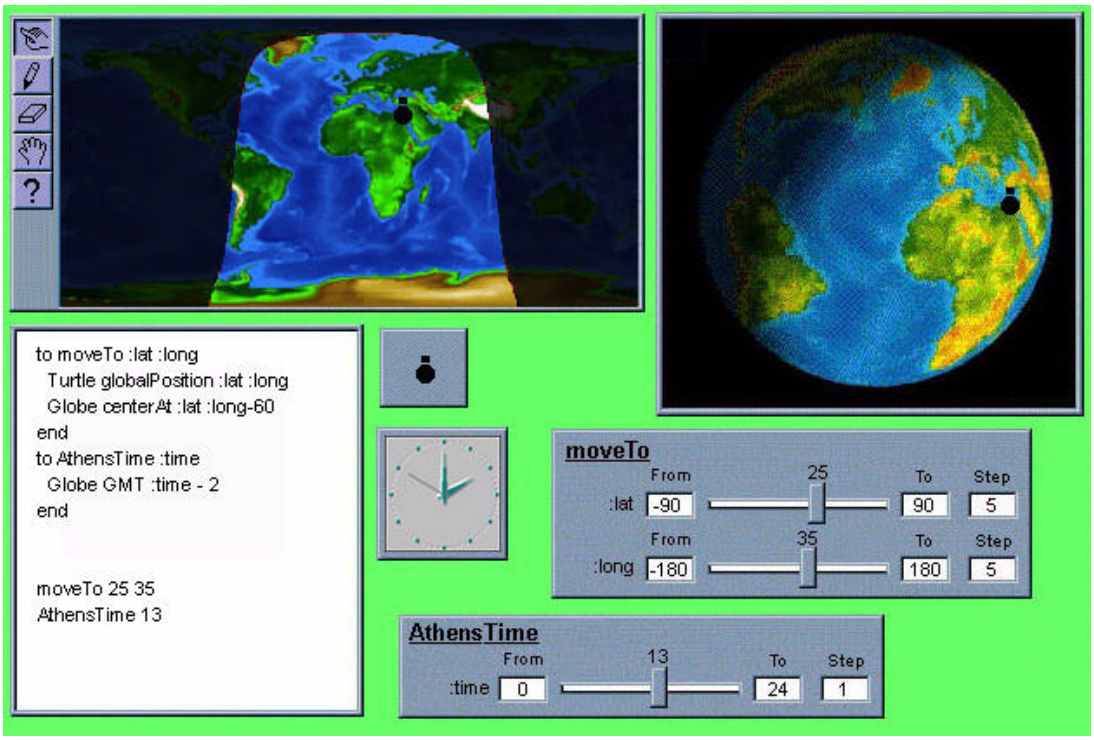

Figure 5. A Geography Microworld with Variation Components

In addition, the Language component now also supports primitives inherited from the newly introduced components. For example, the Globe component provides the "GMT :time" function that adjusts the globe's GMT-time to the specified value affecting the displayed day-night areas, and also the "centerAt :x :y" function that centers the globe's view around the specified location. On the other hand, the Turtle component is now augmented with the "globalPosition :latitude :longitude" function that moves the turtle on the specified position (reflected on the Globe and WorldMap components).

As can be seen in the figure two instances of the Variation component are now applied on two user-defined functions, one that moves the Turtle to a new position and centers the Globe view to a 60´ west offset from that position, and another that converts a given Athens local time to GMT and sets the Globe's corresponding attribute. By sliding each slider pupils can interactively observe how each specific parameter affects the displays of the WorldMap, Globe and Clock and discover the underlying interdependencies.

This microworld provides the ability for simultaneous manipulation of different representations of global space and time, notions that are by no means well comprehended by pupils or adults (Mark and Frank, 1990, Vosniadou and Brewer, 1992). Embedded in the environment, are also the notions of absolute positioning on the earth coordinate system and the way in which time counting is linked to the distance from a fixed longitude. But there is something else which is mathematical in this environment: changes of position, or trips from place to place may be described formally and these descriptions may then be used as objects in higher order constructs, just as in Logo-mathematics.

Conclusions

We have argued that component oriented architecture provides us with the means to construct mathematical software that includes computationally complex objects without excluding formalization as an important feature for user expression. Contrary to common perceptions that technology has moved on from symbolic human computer interaction, formalization can be perceived (because of the new technologies) as a means of expression which can now be better exploited for learning mathematics. For example, we described how high-level building blocks can allow us to do mathematics with turtles on turtle plains or geography maps with the same language and without the need for different pieces of software in each case. In this way it becomes a pedagogical decision, when to use abstract graphical representations of turtle motion and when to embed such motion in other representations including real life simulations. It is also a pedagogical decision when to include formal language in the representational repertoire and when to bypass it by means of alternative representations as in, for example, Noble and Nemirovsky, 1995, Rochelle and Kaput, 1996. Such decisions are no longer either restricted by the reduced functionality offered by all-in-one-box type of software or influenced by the technical overhead needed to use two or more fragmented pieces of software, most often without the required synergy.

However, the task is to define these components at the right level of abstraction so that we can augment the scope of their possible uses. An interesting problem is to what level of generality is it possible or desirable for a component to be defined and how this relates to its specific functionality. For instance how much do maps change if we want to use them beyond geography and geometry? A music map or a history map may need an underlying large database of respective information related to places. Does this make it a map component, or a database component, or two separate components working in synergy and linked by the user?

Mathematical components should be idiosyncratic with respect to other kinds of components. Increasing generality in such a component should not compromise the mathematical integrity in its functionality. It should be defined so that there are no functionality clashes as generality grows and so that a consistent user metaphor (or a small group of tightly linked metaphors) is conveyed. Thus, focusing the argument on mathematical components is an inherently mathematical activity in itself- it invokes us to rethink what is a mathematical object.

The variation tool is such a case, since it has a much wider scope than that of the example we describe here. It can be used for observing the evolution and change in structures dependent on the variable, whether these are represented graphically or numerically. The issues of unsystematic versus systematic and continuous versus discrete variation need to be addressed, as well as the issue of other attributes that can vary apart from numbers or numerical values, such as sound, color and words. The epistemological question of which kind of variation is mathematical and which is not is thus prominent. Apart from the nature of variables, there is also the nature of operations of variables. How could, for instance, the use of the variation component help in understanding the structural properties of binary operations?

Components allow us to embed mathematics in real life or idealized simulations and appreciate its power in applicability but at the same time retain its abstractive power within the same medium and language. Commands may be given to a turtle to stroll around Athens' "Syntagma Square" while a connected "Video" component could display the surrounding view, thus realizing a real-life simulation of the itinerary. The same commands could be part of a Logo procedure describing similar trips in a pure geometrical context by means of components suitable for this domain.

To conclude, developing exploratory software is an expensive and specialized endeavour, requiring different kinds of people to work at the margin of their expertise and interests. Within a component oriented architecture, higher order functionalities designed by engineers under

educational principles can be used by end-users as objects for the creation of exploratory environments. In this way educationalists can focus more on pedagogy in their designs, and at the same time “programmers” can be more effective on their development work.

Acknowledgments

This work is being accomplished in the course of project “YDEES” (http://www.cti.gr/RD3/ydees.html), funded by the European Community Support Framework II (Greek Ministry of Industry Energy and Technology, General Secretariat for R&D, Measure 1.3, # 726.) The project’s objectives include (a) the development of exploratory learning environments for mathematics, geography, physics and music based on a component-oriented software architecture (b) the development of curriculum for focused exploratory activities (microworlds) (c) concurrent development of exploratory classroom cultures in five primary schools (d) teacher education and (e) evaluation of the software, the classroom learning environments and content-specific learning processes.

We would especially like to acknowledge the contribution of the project’s development team for Maths and Geography, who made these ideas come into real software: Petros Kourouniotis, Aggeliki Oikonomou and George Tsironis (Computer Technology Institute colleagues).

Finally, we would like to thank Richard Noss, Celia Hoyles and Al Cuoco for their fruitful comments on drafts of this paper.

Authors affiliations

Chronis Kynigos (ckynigos@atlas.uoa.gr), University of Athens, School of Philosophy, Dept of Phil., Educ. and Psych., Section of Education & Computer Technology Institute, Patras, Greece.

Manolis Koutlis (koutlis@cti.gr), Computer Technology Institute & University of Patras, Computer Engineering Department, Greece.

Thanasis Hadzilacos (thh@cti.gr), Computer Technology Institute, Patras, Greece.

References

Apple Inc. 1996, *OpenDoc User s manual*.

Booth, L.:1984, *Algebra: Children's Strategies and Errors*, NFER-Nelson.

Confrey, J. and Maloney, A.: 1996, "Function Probe", *Communications of the ACM*, August 1996/Vol. 39, No. 8, pp. 86-87.

Cuoco, A.:1995, "Computational Media to Support the Learning and Use of Functions, in Computers and Exploratory Learning", A. diSessa, C. Hoyles and R. Noss (eds.), *Computers and Exploratory Learning*, Springer Verlag NATO ASI Series, pp. 79-108.

diSessa A.A.: 1995, "The Many Faces of a Computational Medium: Teaching the Mathematics of Motion", in Computers and Exploratory Learning", A. diSessa, C. Hoyles and R. Noss (eds.), *Computers and Exploratory Learning*, Springer Verlag NATO ASI Series, pp.337-359.

diSessa, A., Abelson, H.:1986, "BOXER: A Reconstructible Computational Medium." *Communications of the ACM* vol. 29 no 9, pp. 859-868.

diSessa, A.:1993, "Toward an Epistemology of Physics." *Cognition and Instruction*, 10/2-3, 205-225.

Dubinsky, A. and Harel, G.: 1992, "The Nature of the Process Conception of Function", in A. Dubinsky and G. Harel (eds.) *The Concept of Function*, aspects of Epistemology and Pedagogy MAA Notes and Reports Series, pp. 85-106.

Eisenberg, M.:1995, "Creating Software Applications for Children: Some thoughts about Design", in Computers and Exploratory Learning", A. diSessa, C. Hoyles and R. Noss (eds.), *Computers and Exploratory Learning*, Springer Verlag NATO ASI Series, pp.175-196.

Eisenberg, M.:1995, "Programmable Applications. Interpreter Meets Interface". *SIGCHI Bulletin*, 27(2), April 1995, 68-93.

Feurzeig, W. and Richards, J.: 1996, "Function Machines", *Communications of the ACM*, August 1996/Vol. 39, No. 8, pp. 88-90.

Fuys, D., Geddes, D., & Tischler, R.:1985, "An Investigation of the Van Hiele Model of Thinking" in *Geometry Among Adolescents*, New York: Brooklyn College, C.U.N.Y., School of Education.

Hadzilacos, Th., Koutlis, M.: "A framework for the Computer Aided Spatial Education through Geographic Microwords", *Workshop in Advances in Geographic Information Systems*, ACM, Washington DC November 1993.

Hanckock, C.:1995, "The Medium and the Curriculum: Reflections on Transparent Tools and Tacit Mathematics", in Computers and Exploratory Learning", A. diSessa, C. Hoyles and R. Noss (eds.), *Computers and Exploratory Learning*, Springer Verlag NATO ASI Series, pp. 222-240.

Harel, I. and Papert, S.: 1990, "Software Design as a Learning Environment", *Interactive Learning Environments*, 1, 1-32.

Harvey, B.: 1993, "Symbolic Programming vs. Software Engineering - Fun vs. Professionalism - Are These the Same Question?" in P. Georgiadis, G. Gyftodimos, Y. Kotsanis and C. Kynigos, (eds.), *Logo-Like Learning Environments: Reflection and Prospects*, Proceedings of the Fourth European logo Conference, University of Athens, Department of Informatics, August 1993, Athens (Greece), pp. 28-31.

Hoyles, C. and Noss, R.: 1992, *Learning Mathematics and Logo*, MIT Press.

Hoyles C., Healy L. and Sutherland R.: 1991, "Patterns of Discussion between Pupil Pairs in Computer and non-Computer Environments" in *Journal of Computer Assisted Learning*, 7, pp. 210-228.

Hoyles C., Noss R. and Sutherland R.: 1991, "The Microworlds Project: 1986-1989 Final Report to the Economic and Social Research Council". Institute of Education, University of London.

Hoyles, C. and Noss, R.:1991, "Deconstructing Microworlds", in Ferguson, D. L. (ed.) *Advanced Technologies in the Teaching of Mathematics and Science*, Springer Verlag Berlin.

Hoyles, C.: 1991, "Developing Mathematical Knowledge through Microworlds", in A. Bishop, S. Mellin-Olsen and J. van Dormolen (eds.), *Mathematical Knowledge: it s Growth through Teaching*, Dordrecht (Holland), Kluwer Academic Publishers, pp. 147-172.

Jackiw, N.: 1990, "The Geometer Sketchpad", Berkeley, CA, Key Curriculum Press.

Kaput, J., Roschelle, J., 1996, "SimCalc 2nd Year Project Report", SimCalc project.

Koutlis M., Hadzilacos, Th.: 1996, "Avakeeo: the construction kit of geography microworlds", *Second International Symposium on GIS in Higher Education*, NCGIA, Baltimore, September '96.

Kuchemann, D.:1981, "Algebra", in Hart, K. (ed.), *Children's Understanding of Mathematics* 11-16, London, Murray.

Kynigos, C. et. al.:1997, "Microworld design", Interim Report, YDEES project.

Kynigos, C. Gyftodimos, G. and Georgiadis, P.: 1993, "Empowering a Society of Future Users of Information Technology: A Longitudinal Study of Application in Early Education", *European Journal of Information Systems*, vol. 2, no. 2, pp. 139-148.

Kynigos, C., Nikolaides. C. and Oikonomou, A.:1994, "Designing a geometrical microworld based on synergy between direct manipulation and programmability", *Greek Conference for Technological Environments for Education*, Doukas School, 147-155, Athens.

Kynigos, C.: 1995, "Programming as a means of expressing and exploring ideas in a directive educational system: three case studies", in Computers and Exploratory Learning", A. diSessa, C. Hoyles and R. Noss (eds.), *Computers and Exploratory Learning*, Springer Verlag NATO ASI Series, pp. 399-420.

Kynigos, C.: 1997, "Dynamic representations of angle with a Logo-based variation tool: a case study", *Proceedings of the Sixth European Logo Conference,* Budapest.

Lave, J.:1988, *Cognition in Practice*. Cambridge, Cambridge University Press.

Mark, D., Frank, A.: 1990, "Experiential and Formal Models of Geographic Space", *NCGIA Technical Report* 90-10.

Microsoft Inc. 1996, *OLE User s manual*.

Noble, T. and Numirovsky, R.: 1995, "Graphs that go Backwards", in L. Meira and D. Carraher, (eds.), *Proceedings of the Nineteenth International Conference for the Psychology of Mathematics Education*, Recife, Brazil, vol. 2, pp.256-263.

Noss, R. (in press): "Meaning Mathematically with Computers", in Bryant P. & Nunes T. (eds.) *Children doing mathematics,* C.U.P.

Noss, R. and Hoyles, C.:1996, *Windows on Mathematical Meanings*, Kluwer Academic Publishers, Dordrecht/Boston/London.

Resnick M.: 1994, *Turtles, Trinities and Traffic Jams: Explorations in Massively Parallel Microworlds*, Cambridge, MA, MIT Press.

Resnick M.: 1995, “New Paradigms for Computing, New Paradigms for Thinking, Computers and Exploratory Learning” in Computers and Exploratory Learning”, A. diSessa, C. Hoyles and R. Noss (eds.), *Computers and Exploratory Learning*, Springer Verlag NATO ASI Series, pp. 31-44.

Roschelle, J., Kaput, J.J.: 1996, “SimCalc Mathworlds for the Mathematics of Change”, *Communications of the ACM*, August 1996/Vol. 39, No. 8, pp. 97-99.

Roschelle, J., Spoher, J.: 1997, *Educationl Object Economy*, http://trp.research.apple.com/EdEconomy/index.html.

Sendova, E. and Sendov, B.:1993, “Learning to Speak Mathematically and Speaking Mathematically to Learn”, in C. Kynigos et al. (eds.), *Proceedings of the 4th European Logo Conference*, Doukas School Publication, pp. 281-289.

Sinclair, H.: 1987, “Constructivism and the Psychology of Mathematics”, *Proceedings of the Eleventh International Conference for the Psychology of Mathematics Education*, pp. 28-42.

Sutherland, R.: 1989, “Providing a Computer Based Framework for Algebraic Thinking”, *Educational Studies in Mathematics*, 20, Kluwer Academic Publishers, the Netherlands, pp. 317-344.

TERC Inc:1993, Curriculum Guide for the Tabletop Senior and the Tabletop Junior.

Vergnaud, G.:1987, “About Constructivism”, *Proceedings of the Mathematics Education*, pp. 42-55.

Vosniadou, S., Brewer, W.F.: 1992, “Mental Models of the Earth: A Study of Conceptual Change in Childhood”, *Cognitive Psychology*, 24, pp. 535-585.

Wolfram Research Inc., 1996, “Mathematica user’s manual”.