E-Slate: a kit of educational components

Manolis Koutlis

koutlis@cti.gr

George Birbilis
Margarita Dekoli
Thanasis Hadzilacos

Chronis Kynigos[1]
Kriton Kyrimis
Xenia Siouti

George Tsironis
George Vasiliou

Computer Technology Institute
Kolokotroni 3, 26221
Patras, Greece
+30 61 273496

Abstract

This paper reports on research work accomplished during the last five years on the design and development of E-Slate, an end-user environment for creating high-quality educational software of exploratory nature which, among other things, can take full advantage of the internet/web. Educational activity ideas can be turned into software by anyone having elementary web-authoring skills in the form of interactive web-pages, that is, pages containing specially designed educational *components*. These components, which are provided as a kit of pre-fabricated computational objects, can be very easily "plugged" together in any desired configuration so that they collaborate and attain additive functionality. In addition, the behavior of both components and activities as a whole, can be "programmed" through a symbolic Logo-based language. E-Slate is currently based on the Java platform and related technologies, and hosted at http://E-Slate.cti.gr.

Keywords: component software architectures, educational software development, authoring systems, exploratory software, Internet based tools.

[1] Also affiliated with the University of Athens

Background

It is recognized that quality educational software is very expensive to develop, hard to find around, and usually evolving in highly specialized research labs [8]. This is not without a reason: while the most effective designers of software with real educational value come from the broader educational community (curriculum designers, teachers, pedagogists, pupils), they are rarely -if ever- involved in the development process which is dominated by technicians or publishers. Besides, educational theories and didactic approaches vary so widely -often being contradictive- that educational software constitutes a large mosaic of diverse design philosophies which cannot be accommodated into a simple framework of software tools for the -typically- computer-illiterate audience.

Put in another way, while the educational community *should* be in the steering wheel of courseware production, the roads that have to be crossed require formula-1 expertise. Do they have to acquire such skills? Is it better to hire a driver? Or should they look for better cars? Let's see how educational software is typically developed today.

1. A teacher that has some nice design ideas hires a programmer (or vice versa), describes the requirements and the programmer comes back with the program. The circle repeats for future ideas. The results could be nice pieces of software, but otherwise unreusable and structurally or programmatically inaccessible to end-users, requiring unacceptable resources for even the slightest modification.
2. Teachers adopt one of the so-called authoring tools and set the target of developing the program themselves. At the end of the day and after they find themselves spending much more effort on building what is to be studied than really studying it, they usually come up with mediocre constructions. Disappointed they turn again to a specialist to finish up the job, now insisting that the programmer uses that same tool of their choice, as in that way they could later on make the minor but necessary for the teaching style interventions. The programmer reluctantly agrees, only to find himself in a cumbersome environment that cannot satisfy the development requirements, as it was not design to do so. The end products are strongly bound by the underlying tool's capabilities.

This is the situation where E-Slate comes to contribute: by adopting a component-oriented development approach [4, 6, 7] it offers two distinct faces, one to the programmer and a different one to the end-user allowing each one to work effectively. The courseware designer is provided with a kit of reusable, prefabricated parts (components), together with a "glue" mechanism that allows the interconnection of any number of required components into fully functional constructions. Programmers constantly expand the library of components by developing new entities as required by specific educational needs -utilizing their preferred software engineering methods.

By exploiting the special characteristics of a curriculum field, E-Slate provides components that encapsulate domain-specific features which are familiar to the target audience. End-users work on a high conceptual level manipulating known entities with expected behaviors. In addition, E-Slate provides more than one ways of expression: visual through direct manipulation, and symbolic/algorithmic through a Logo-based programming language.

Editable applications

In an environment like E-Slate, ideas on educational activities can be turned into full-featured software in minutes. All that is needed is basic literacy on web-page design. With simple drag&drop operations, users can edit a page like the one shown below to setup an experimental microworld on vectors[2] in which pupils get acquainted with the relevant concepts through a game-like activity of navigating airplanes around Europe. The Map, Vector, Airplane, Clock and MasterClock components (Java applets in this implementation) are laid out on the page and suitably interconnected so that they work in concert (see next chapter): the top vector controls the plane's thrust and direction, the middle vector controls the air speed, while the bottom vector displays the calculated air-speed of the plane (roles could be assigned to the three Vector components as required). The Plane component is connected to the Map component, while the MasterClock component provides ticks to the whole simulation.

[2] The specific activity was designed in the context of project IMEL (see aknowledegments for a full reference) by Prof. R. Noss and prof. C. Hoyles, University of London, Institute of Education.

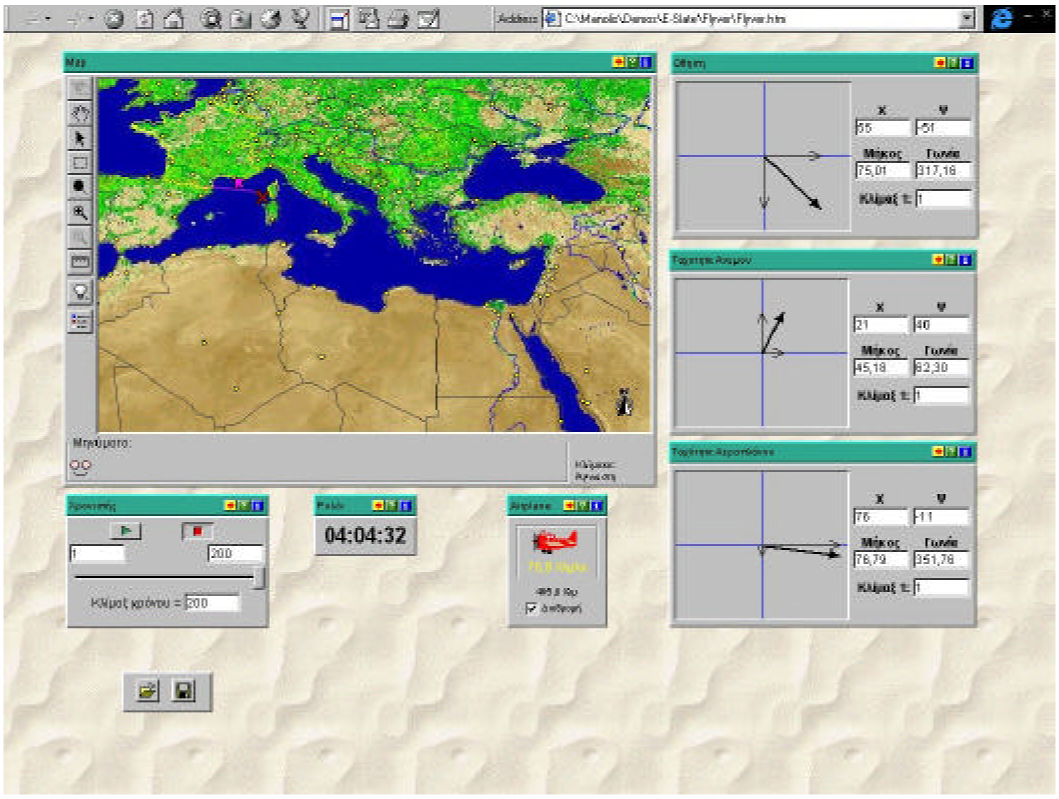


Figure 1. Learning about vectors through flight simulations around Europe

After playing around for a while the idea comes, that it would be nice to have two maps instead of just one, so that the plane's flight could be observed in different map zoom levels at the same time, and observations on the notion of geographic scale could be realized. Again, in just a few minutes, this is doable: teachers can copy/paste the existing Map component (so that two instances appear on the page), change the layout a bit and the new idea is ready to go.

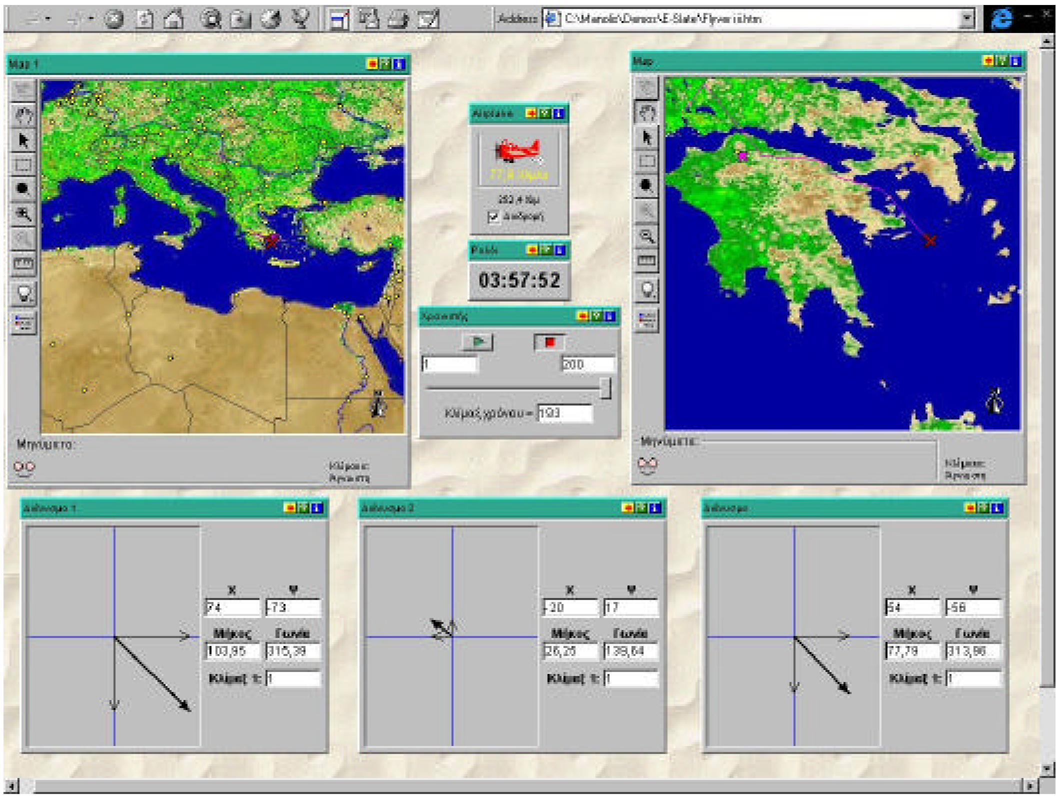
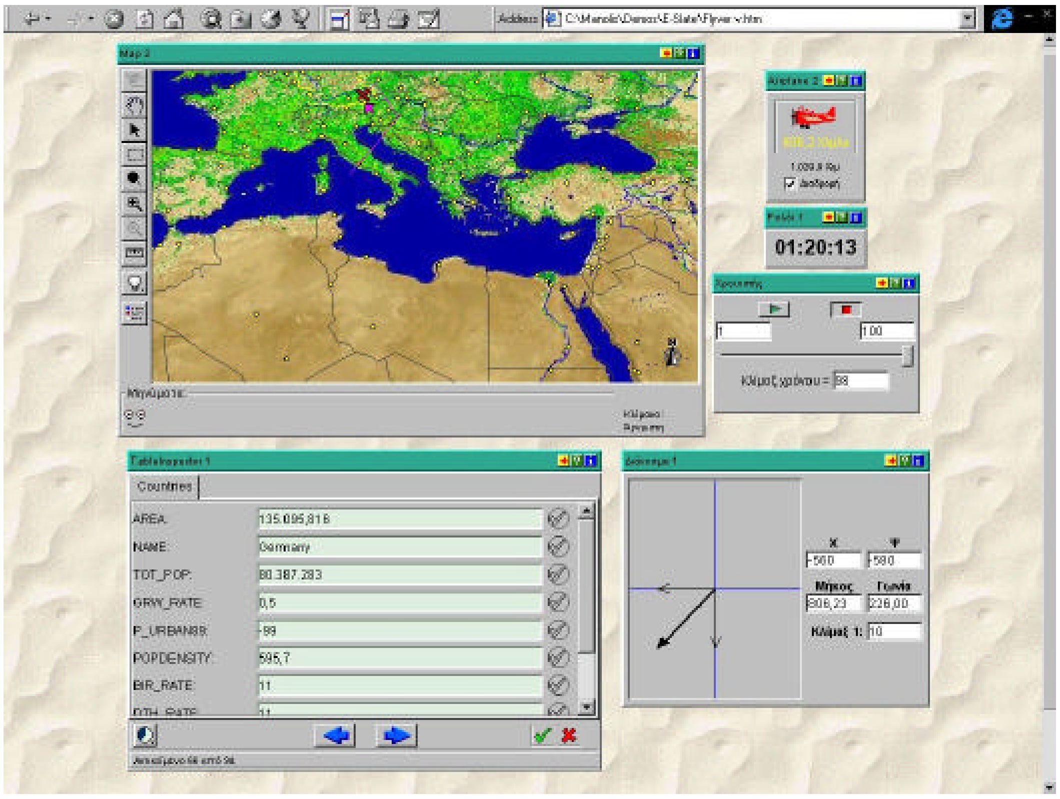


Figure 2. Experimenting with the notion of geographic scale.

Finally teachers spend another five minutes to realize yet another idea: as the plane passes above European countries and cities, it retrieves the corresponding records of the map's underlying database and routes them into a record template component (bottom left) so that features like name, area, population, etc can be observed and further studied in a Geography course.



AREA:	135.095,818
NAME:	Germany
TOT_POP:	80.387.283
GRW_RATE:	0,5
P_URBAN88:	89
POPDENSITY:	595,7
BIR_RATE:	11
DTH_RATE:	11

Figure 3. Exploring the European countries.

Seamless inter-component synergy

How is the functionality of a microworld defined? Each component carries a number of colored "plugs" (of the shape of puzzle-like pieces) which appear on demand by clicking on the ➡ button on a component's top bar. Each plug has a label that describes its role (for example "Color", "Finding", "Photo", etc). Plugs can be plugged one into the other by simple point-and-click operations. In that way, information flows from one component to another and synchronized behavior is achieved. The plugging rule is straightforward: only plugs of the same color and complementary shape (male-female) can be plugged. Plugging, is not a separate mode from the components' normal operation; it can be performed in parallel and the effects can be observed immediately.

The figure below depicts the process of defining inter-component associations in a history scenario where pupils are assigned the role of an archaeologist, trying to recreated the Mycenean civilization by excavate sites and observing the artifacts[3]. In this snapshot, the (male) plug "Photo" of the Template component (bottom-left, displaying the Archeological Findings table which is fed from the Map component) is plugged into the corresponding (female) plug of the Picture Frame component (top left), so that the a finding's picture is displayed on the frame whenever the particular finding is selected (excavated) by the hoe tool on the map.

[3] The specific activity was designed in the context of project Odysseus (see aknowledgements for a full reference) by E. Gika, Greek Pedagogical Institute.

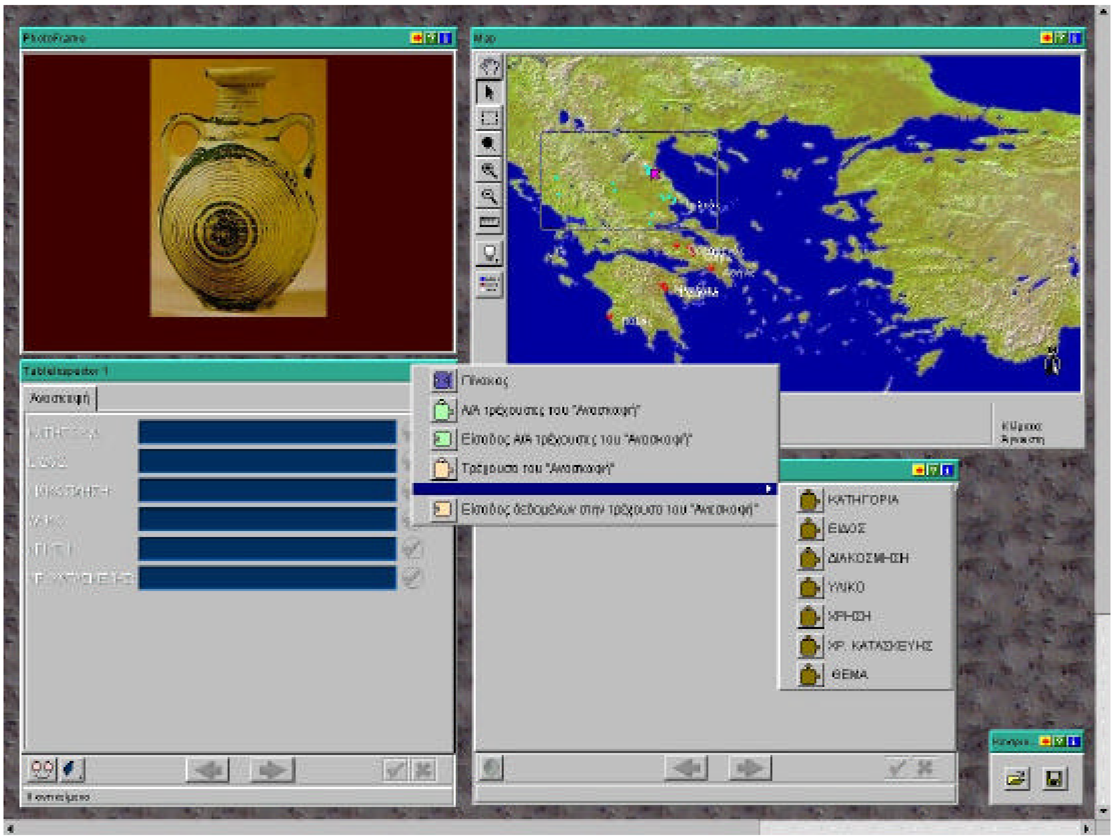

Πίνακας
Α/Α τρέχουσας του "Ανασκαφή"
Είσοδος Α/Α τρέχουσας του "Ανασκαφή"
Τρέχουσα του "Ανασκαφή"
Είσοδος δεδομένων στην τρέχουσα του "Ανασκαφή"
ΚΑΤΗΓΟΡΙΑ
ΕΙΔΟΣ
ΔΙΑΚΟΣΜΗΣΗ
ΥΛΙΚΟ
ΧΡΗΣΗ
ΧΡ. ΚΑΤΑΣΚΕΥΗΣ
ΘΕΜΑ

Figure 4. The inter-component "wiring" process.

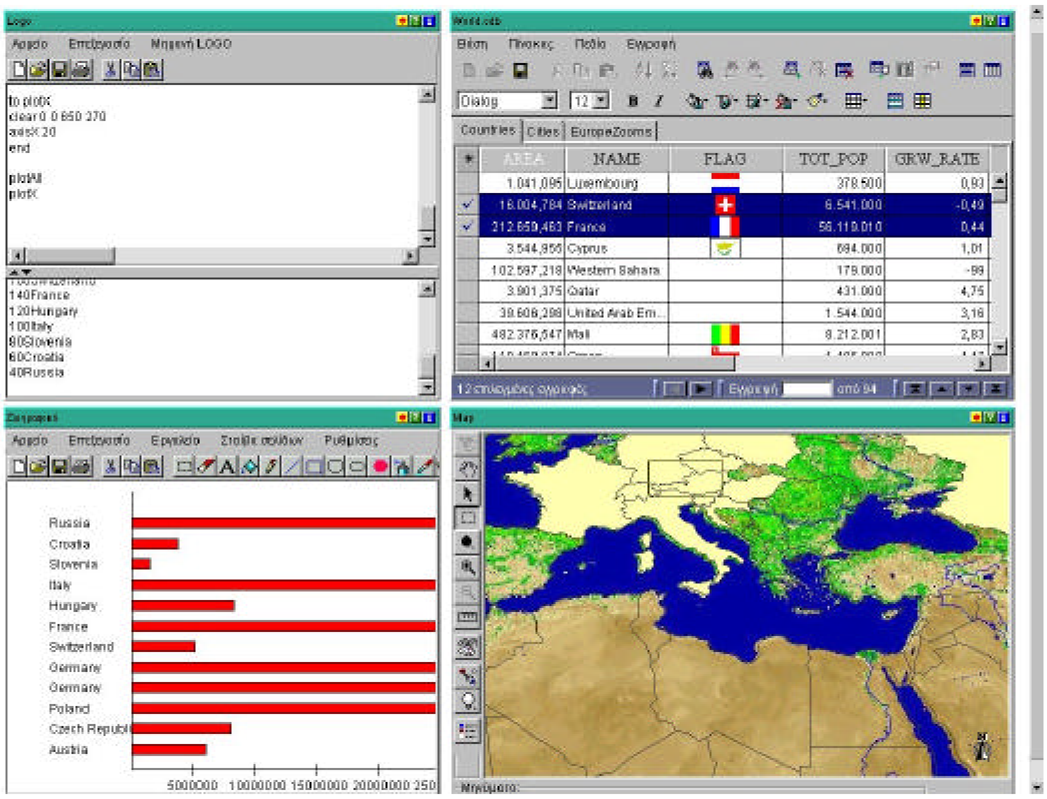
Programmability power

In addition to their ability for establishing inter-component webs through the plugging mechanism as described above, E-Slate components are accessible from within a Logo-based programming language through suitably defined primitives. In that way, components are available as high level objects to the Logo language amenable to introspection and programming of their behavior, or vice versa, E-Slate Logo[4] is their *scripting* language.

Each component carries its own domain-specific primitives and dynamically registers them to the scripting mechanism during initialization. Once registered, primitives can be issued in the form of statements and commands to the Logo interpreter and executed by the parent component. Statements like TELL, ASK and EACH define the targeted (set of) components identifying them by name [2].

In the example depicted below, the Database component displays (edits) the countries and cities database tables which are fed by the Map component (through plugging). As we graphically select an area on the Map, the corresponding database records are also selected in the Database component, (blue lines) and vice versa. Using the Logo editor/interpreter component (top left) we define a couple of primitives that retrieve the selected set of records from the Database component and draw a population plot based on the corresponding values for each of these countries. To do so, the script uses primitives that were registered by the Database component (returning a list of the selected records, the values of specific cells, etc), and by the Canvas component (bottom left, drawing lines, shapes, strings, etc on the canvas pages).

[4] E-Slate Logo is an extension of the Logo language to serve as a components' scripting mechanism. The current implementation is based on TurtleTracks, a Java-based Logo interpeter [1].



```
to plotX
clear 0 0 650 270
axisX 20
end

plotAll
plotX
```

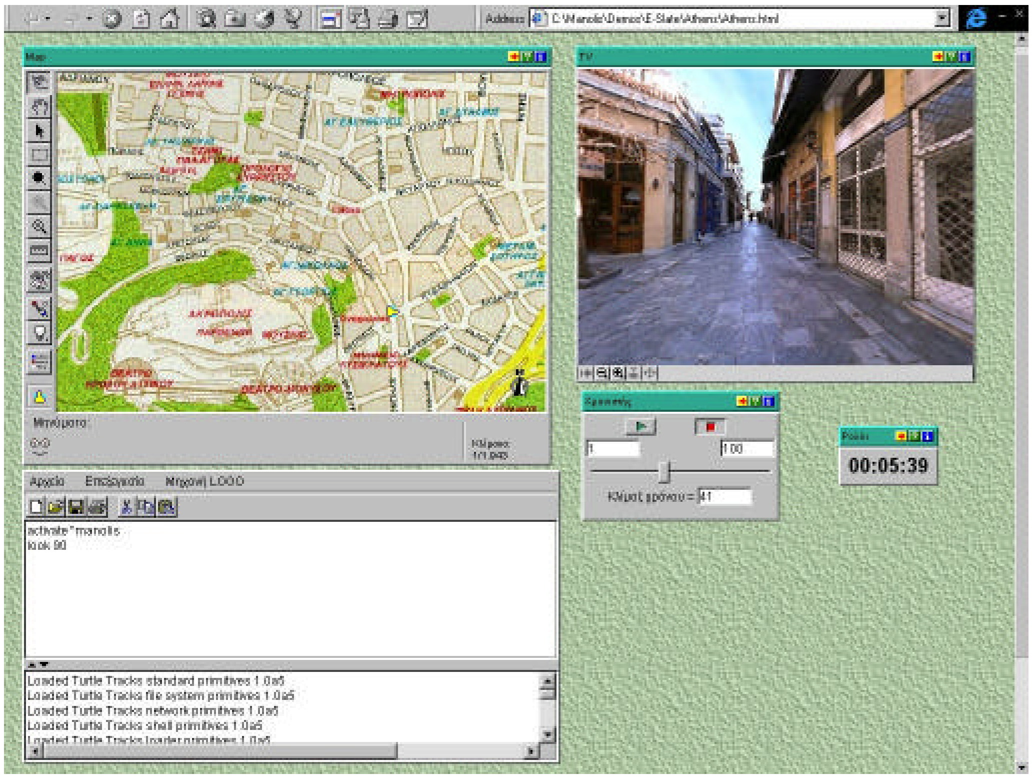
```
140France
120Hungary
100Italy
80Slovenia
60Croatia
40Russia
```

	AREA	NAME	FLAG	TOT_POP	GRW_RATE
	1.041,095	Luxembourg		378.500	0,93
✓	16.004,784	Switzerland		6.541.000	-0,49
✓	212.659,463	France		56.119.010	0,44
	3.544,956	Cyprus		694.000	1,01
	102.597,218	Western Sahara		179.000	-99
	3.901,375	Qatar		431.000	4,75
	30.608,298	United Arab Em...		1.544.000	3,16
	482.376,647	Mali		8.212.001	2,83

12 επιλεγμένες εγγραφές   Εγγραφή   από 94

Figure 5. Logo scripting orchestrates the microworld's behavior.

In another example depicted below Logo scripts are used to navigate anthropomorphic agents around the center of Athens, by means of spatial and navigational commands in a game-like activity (search of a lost treasure). As they go along, agents report whatever they find, see or stumble on their path to other components for further viewing or manipulation (in the example, the Picture Frame component displays an agent's view).



```
activate "manolis
look 90
```

```
Loaded Turtle Tracks standard primitives 1.0a5
Loaded Turtle Tracks file system primitives 1.0a5
Loaded Turtle Tracks network primitives 1.0a5
Loaded Turtle Tracks shell primitives 1.0a5
Loaded Turtle Tracks loader primitives 1.0a5
```

Figure 6. Agents navigated by Logo commands.

Benefits and perspectives

Our experience with developing and using E-Slate demonstrated some clear benefits of the component oriented approach in synthesizing educational software over traditional methods…

- End-users think in terms of high level building blocks which represent entities at the level of their conceptual domain, instead of dealing with primitive data structures, algorithms and programming language specifics.
- The flexibility offered for editing component configurations in virtually any form, opens new ways of thinking, new possibilities for combining behaviors at will that were not possible before.
- Large-scale reusability is made possible, which means great reduction of development resources (develop once, use many).
- The roles of educators and technicians while developing educational software are more clearly defined, and the involvement more balanced. Each side can make the right decisions on its own: educators can decide on the teaching approach, exact content, learning objectives, while technicians can decide on the tools, APIs and algorithms that are most suitable for the task at hand.
- Components developed by separate teams, if designed properly, can interoperate and add to a common pool of reusable components. In that way, a developer's work gains additive value (since it combines with others' work) and users have the option to choose among a potentially large gallery of functionalities (the equivalent of choosing various parts for a home-stereo system regardless of manufacturer).

…but also raised tough design problems and identified challenging research issues (see also [5]).

- Designing reusable and interoperable components is far more difficult than designing standalone applications. Decision on the components' granularity is a crucial and tough job: which part of the required functionality goes into which component, so that it is as reusable as possible while remaining at a high conceptual level as a comprehensible entity to the user.
- Technological standards for supporting large scale component oriented development (a prerequisite for achieving inter-team collaboration) have just recently emerged and are not considered mature enough yet (for example, JavaBeans™, InfoBus™, etc).

Status

Currently, E-Slate is developed with Java and related technologies and is freely available for use and evaluation from http://E-Slate.cti.gr.

Acknowledgements

E-Slate has been funded by the following projects:

- Project "YDEES" (EU Support Framework II, Greek Ministry of Industry Energy and Technology, General secretariat for R&D, Measure 1.3, Project 726) (1995-98)
- Project "IMEL", EC DGXXII, Socrates 25136-CP-1-96-1-GR-ODL (1996-98)
- Project "ODYSSEAS" (EU Support Framework II, Greek Ministry of National Education and Religious Affairs) (1997-99)

References

1. D. Azuma TurtleTracks documentation, http://www.ugcs.caltech.edu/~dazuma/turtle/
2. M. Koutlis, Ch. Kynigos, A. Oikonomou, G. Tsironis, "Empowering Logo through a Component-Oriented Approach", 7th Eurologo Conference, Boudapest, Hungary, 1997.
3. M. Koutlis, P. Kourouniotis, K. Kyrimis, N. Renieri, "Inter-component communication as a vehicle towards end-user modeling", *ICSE'98 Workshop on Component-Based Software Engineering,* Kyoto, Japan, 1998.

4. C. Kynigos, M. Koutlis, Th. Hadzilacos, "Mathematics with Component Oriented Exploratory Software", *International Journal of Computers in Mathematics Education*, Kluwer Academic, Vol 2, 1997, pp. 229-250.
5. J. Roschelle, M. Koutlis, A. Reppening, "Lessons from Research with Educational Software Components", sumitted to IEEE Computer in response to a call for articles for a special issue on component software (June 98).
6. J. Roschelle and J. Kaput, "Educational software architecture and systemic impact: The promise of component software," *Journal of Educational Computing Research*, 14(3), 1996, pp. 217-228.
7. A. diSessa, “Educational Toolsets”, 21st conference of Psychology of Mathematics Education (PME) 21, Lhati Finland 1997.
8. E. Soloway, "No One Is Making Money in Educational Software", Communcations of the ACM, February 1998.