

META: Multimedia Environment for Training Applications

David Luigi Fuschi, Vincenzo Piuri
Department of Electronics and Information
Politecnico di Milano
Piazza L. Da Vinci 32, 20133 Milano, Italy

ABSTRACT

In present days there are no longer *jobs for life*: continuous training is a necessity and workers will often require a multitude of skills. Training must therefore evolve to become a more pro-active process of self-education. Distance learning and self-instruction systems can meet the demand that arises from the rapid obsolescence of knowledge and the need for lifelong education. According to surveys, companies that have switched from traditional training to multimedia tools based one have saved a great deal of money and time.

META project starting point was the need to overcome the present drawback of multimedia products development: *its unpredictable length*. This mainly depends on raw material and authoring packages quality, authoring packages flexibility and response to courseware increasing complexity. Existing products are mainly oriented either to presentation or to CBT-like applications. In both cases, available tools are not properly suited to flexibly respond to all needs of courseware development. In addition almost no tools allow to derive both stand-alone and distributed products as well as distributed tutoring. META authoring system tries to present a possible solution.

1. INTRODUCTION

Companies are discovering the many advantages of multimedia training when compared to traditional teaching. The novel approach allows quality training with considerable time and money saving.

Packages supporting multimedia products development can be classified in two main families: *authoring systems* and *presentation systems*. Among the available ones, *Authorware Professional*, *IconAuthor*, *ToolBook*, *Animation Works Interactive*, *Ultimedia Builder/2*, *HSC Interactive*, *Hypermedia Professional*, *Director*, *Astound*, *Super Show and Tell*, *Q-Media*, *Cinematic*, *IM-Age*, *Action* and *Multimedia Viewer Publishing Toolkit* are the most powerful and commonly used. Each of them has its own points of strength and of weakness. Most of them are very expensive, some are

too complex for an easy usage by non skilled personnel, some are suited for complex presentation but do not allow a real hypertextual navigation, some are not flexible enough, some put constraints on the available data type. Most of them are mainly for stand alone applications production.

META ESPRIT project 8045 aims to integrate a broad range of consolidated and emerging technologies into a common framework supporting development and fruition of distributed training products and services for both business and home markets. The project is developing a professional authoring environment supporting: *publishers* in producing courseware, *teachers* in personalising and delivering lessons to students, *tutors* in managing real time interaction and control on students' work and *students* and *professionals* in their continuous training process. These different characters correspond to different roles in the courseware development / fruition cycle. A *publisher* professionally develops courseware relaying on expertise and expects the most out of an authoring product even when some usage complexity has to be accepted. A *teacher* usually wants to get the most out of his pupils, therefore wants to adapt his teaching tools to theirs needs with the least effort. A *tutor* is usually engaged only into support activities, even to large number of customs and therefore needs an easy tool with large support and control capabilities. A *student / professional* wants an easy to use and pleasant learning tool possibly not constrained by a specific hardware platform, where remote support can be attained in an easy and effective way. The basic idea is that a better development / fruition environment could be derived integrating available technology under direct control of *publishers* and *end-users*. This leads to a more balanced system. META framework includes interactive authoring tools for easy courseware development / customisation, remote interaction capabilities encompassing text, voice, image, video, still and moving image handling features based on high compression techniques, high quality voice / sound sampling and compression and recording (4 to 8 kbps), colour video transmission over ISDN or LAN connections (12.5 to 25 frames 320x256 pixels per second), a multimedia database holding courseware structure and components, a lesson delivery system, learner's activities logging capabilities with automatic scoring features. The expected commercial project

results are consumer stand alone CD-ROM products, networked services and training courses. The measure of project success will be how good a balance it will be achieved between users' potentially unlimited communication requirements and the limited capacity of currently available communication technologies.

An initial study identified market opportunities, defined basic required components and existing technologies, products and standards matching the requirements. On this basis the overall architecture was defined, a proper interfacing / integration plan was completed and items to be developed were specified. Network and distributed software operation were modelled foreseeing both a strictly and a loosely connected version. As learning system are very demanding, motion video, still images and audio compression / decompression techniques were analysed and integrated in order to minimise storage requirements. This approach allows minimising communication costs. Particular attention was placed in defining support for the tutor to end user co-operative interworking.

Major project milestones have been market and competitors analysis, component and architecture definition and components development. System components integration,

framework implementation and real life applications testing are still under development. Project development has been carried out using OO techniques, while code development has been carried out in C/C++ under Windows 95, Windows 3.11 and Acorn-RISC/OS.

2. THE META ENVIRONMENT

META system is based on several subsystems, each functionally characterised to support author, tutor and user activities (see Fig. 1). As one of the project constraints is portability under various platforms, a object oriented solution has been adopted. Moreover, as this is an integration project, all non-essential components have been chosen among existing ones and integrated into the system by building up an appropriate interface layer.

The object oriented approach has also naturally lead to a layered structure, where the foreseen layers are:

- the *flow layer*, supporting curricula definition through lesson modules linking and scoring based path definition as well as fruition of lessons in the student environment,
- the *tree layer*, defining hypermedia links among lesson

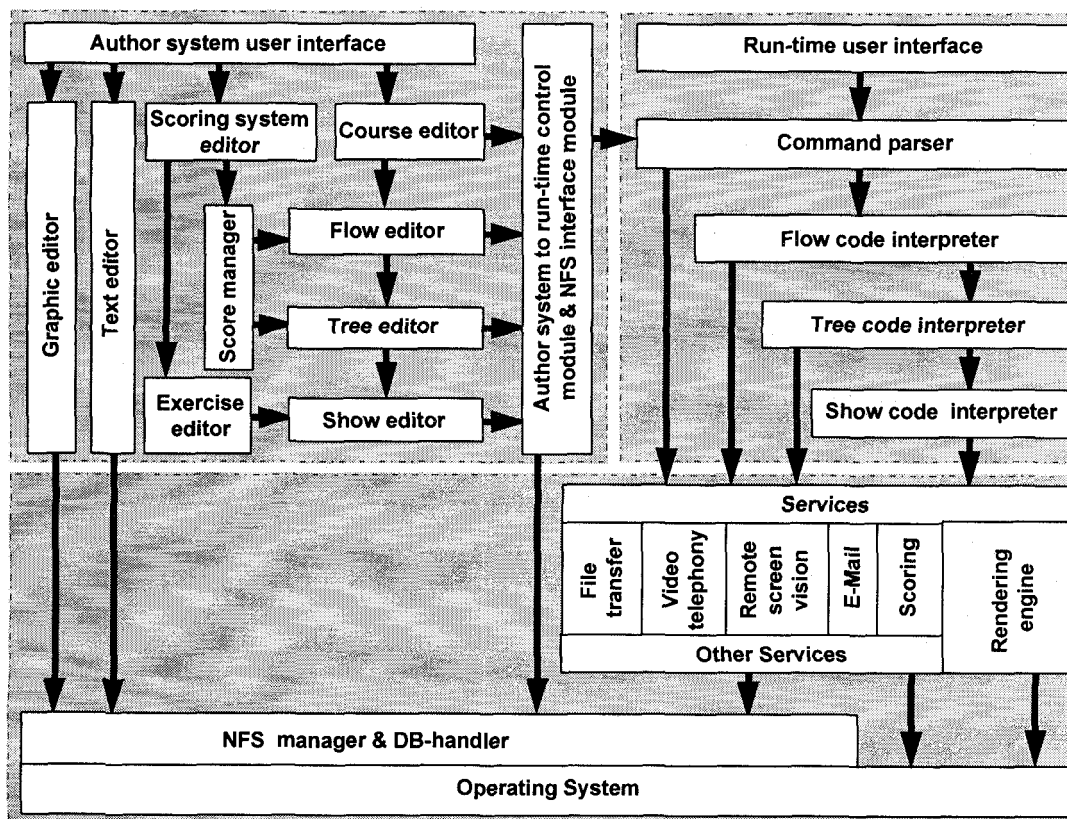


Figure 1 - The META Environment

frames as well as fruition of the lessons in the student environment,

- the *show layer*, creating media frames and interacts with the *GUI* as well as fruition of the lesson components in the student environment.

In order to ensure maximum flexibility a course has been considered as composed of modules, each composed of one or more lessons. Each lesson is in turn composed of one or more *pages*. Each page can be considered as a presentation. Therefore each layer is in relation with a specific topic: *flow* with *courses / curricula*, *tree* with *modules / lessons* and *show* with *pages*. This will be clearer examining in detail the different roles played by each layer.

3. THE LAYERED STRUCTURE

The *Flow layer* is the top layer. It deals with the definition of courses / curricula through lesson and modules linking. The knowledge path is defined and conditioned by scoring results. That is the author places at this level conditions on variables, related to scoring, which are used to determine the learner's knowledge path. Tools for defining structures and templates, in which to fit prepared modules or lessons, are available at this level. There is also a very broad set of conditions and rules to be used in order to shape the overall learning process. On the other hand there are no display or presentation capability at this level. Courseware customisation can be easily performed at this level without modifying courseware content.

The *Tree layer* is the intermediate one. It deals with the internal structure of modules and lessons. Each module / lesson is seen as a tree structure strictly connected with the navigation information and deeply conditioned by user interaction. All possible user's choices are defined at this level. The resulting tree structure can be very complex as hyper-links are available. A very broad set of controls are available in order to correctly and safely design the navigation structure. On the other hand there are almost no display capability at this level. A defined tree structure can be replicated to allow management of various languages. Such procedure requires only to change those presentation items which are language dependent. Therefore it is easier to prepare multi-language training products.

The *Show layer* is the lowest one. It deals with the sequential presentation of a series of screens, images, videos and all that the author wants to be viewed as a continuous elementary unit. In this layer is allowed only a very limited set of instruction for navigation control. Variables can be set during a presentation. On the other side in this layer is available a huge set of effects and feature which are related to presentation itself. Transition effects, picture loading effects, text presentation effects and many other are available at this level. This layer interacts directly and deeply with the operating system (OS) through the *Service interface layer*.

The overall system structure can be split into three main components: an *Authoring package* (area in the upper left part of the picture), a *Runtime environment* (the area in the upper right part of the picture) the *OS and services* (the lower part of the picture). A *Scoring sub-system* internally completes authoring activities and allows to save runtime exercise related information. For the final user (student or professional) attending a series of lessons, the only relevant sub-system is the runtime one. The authoring package is viewed as an add-on sub-system which is connected with the runtime one via a set of specific interfaces. The core of the whole system is the *Runtime environment* composed of a *Runtime user interface*, a *Command parser* and the various *Code interpreters*.

The adopted layered approach is evident examining this set of subsystems. The user issues a command to the system by interacting with the *Runtime user interface* via a mouse click or an input field. His interaction is examined by the *Command parser* who detects the actual destination of the received input and forwards it to the final destination (service or code interpreter). Such an approach is possible as each used function and system service has a correspondent set of calls. Each *Code interpreter* interprets and executes a specific code, where there can be a call to a different layer, naturally a lower one. Therefore it executes directly all statements belonging to his own level and forwards, to the lower one, all commands that can't be directly executed.

4. AUTHORING SYSTEM TOOLS

The authoring tool suite allows to prepare a courseware either starting from scratch or from an existing one that needs to be customised. The overall environment and the sequence of activities are presented as a workbench in Fig. 2. In the authoring environment the *Graphic editor* and the *Text editor* are common tools, already present in the OS platform, which have been integrated into the overall authoring environment. They will be accessible via a specific *Authoring user interface* developed for the project. The *Authoring environment* is, in turn, composed of other component.

The *Scoring system editor* allows to design the overall structure that controls the scoring activity in the produced courseware. That is, it allows to produce exercise templates and scoring rules that will be followed at runtime by the scoring system to condition / affect the user knowledge path. This module interacts with both the *Score manager* and the *Exercise editor*. The first allows the author to state the actual scoring system behaviour while producing a course. It allows to place in the *flow* or *tree* codes all the necessary source code for controlling the user learning process, by inserting exercises and branch points related to exercise results. Such module may affect both the general navigation process at *flow* level as well as the module internal flow at *tree* level.

The *Exercise editor* allows to prepare new exercises and to connect, or modify, already existing ones. This module interacts only at *show* level as it does not directly affect in any manner the navigation.

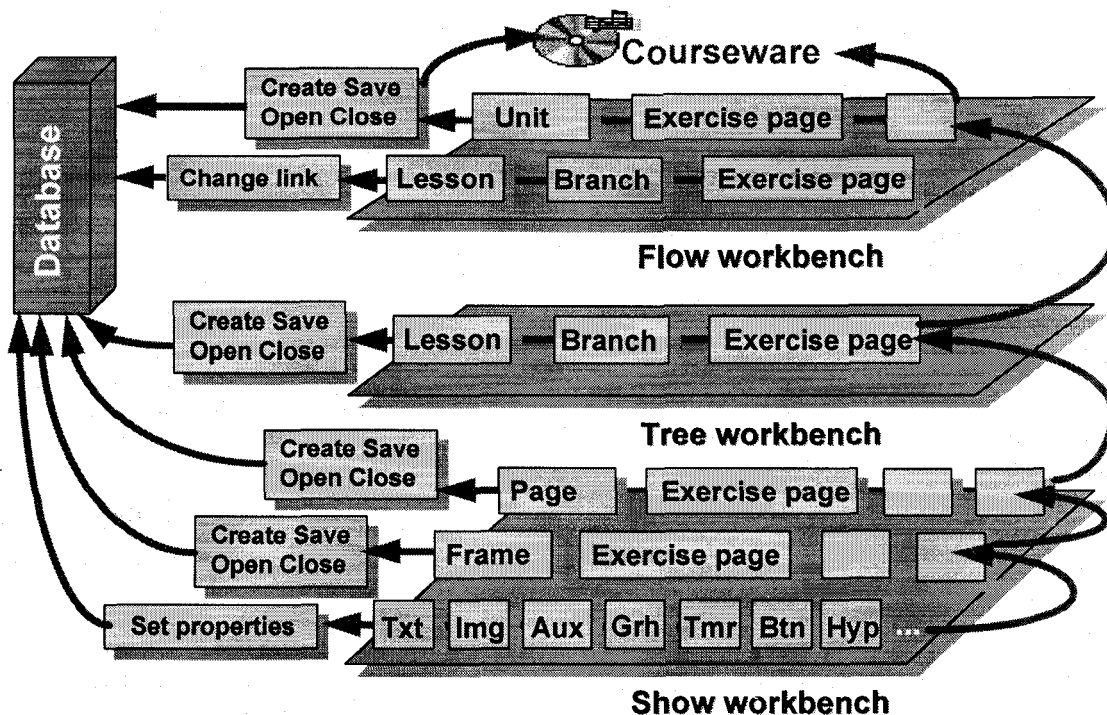


Figure 2 - The META Workbenches

The *Course editor* is the authoring tool used by the author in order to prepare a complete course. It allows the author to sketch the overall structure in terms of organisation. This module interacts directly with the *Flow editor* and is meant, essentially, as a tool giving the author the chance to plan and document courseware development, to organise courseware structure, interface, target (stand-alone or networked fruition) and to see how the course will run while still under development.

The *Flow editor* allows the author to prepare, via graphic tools, the courseware structure in terms of modules and lessons. The author can use various tools to define the runtime functionality of the user interface. That is, at this level are stated the consequences, in terms of navigation, of each user interaction (starting from a mouse click on a button, up to the score-driven modification to navigation paths). Most of the *Score manager module* commands results are also available at this level, in order to plan navigation changes due to user's scoring results. This module is intended mainly for courseware architecture drawing and modification. It can be used for customisation of an existing courseware without affecting the content of the course itself. The author can adapt

navigation and learning paths to the end user requirements without having to modify directly the lower level codes.

The *Tree editor* gives the author the ability to structure the various topics presentation. Moreover it allows to set system reaction to user choices. Operations to be performed when a button is pressed are defined at this level, as well as the way of treating pressed / un-pressed buttons aspect at user interface level. How system services will be accessible to end-users is also stated at this level. It is here that *show* level codes are linked to the chosen course structure. The overall functionality of the courseware is implemented at this level. Overlapped tree structure can be defined in order to deal with multiple language courses. Also the information and controls coming from the *Score manager module* are integrated at this level. That is: navigation can also be driven by score results in accordance with the rules defined with the *Score manager module* and integrated into the *flow* level. In certain circumstances, for example, the author may will the user to follow a specific path if score falls below a certain level. The level may be defined / modified at *flow* level, the decision rule while sketching the scoring system and the actual system behaviour at *tree* level by using the available tools as

condition evaluation, variable settings and jump instructions. This module is the core of the courseware development as it links presentations together with the interface by defining both the logic and the way things work at runtime.

The *Show editor* allows the author to prepare the actual appearance of each page / lesson. Back wall pictures can be picked up and used by applying displaying effects. Buttons appearance and related text can be defined. Videos and audio can be recalled as well as animation and text displayed. In all steps of courseware development the author has the ability to preview the actual results by running the code as a user without leaving the authoring environment. Moreover the author has the capability to trace his work in order to verify, step by step, its correctness. For such a reason is foreseen a special interface layer that connects the authoring environment to the runtime one. Such interface layer allows the author to exploit all functionality available in the runtime environment as if they were part of the authoring one.

Therefore using the aforementioned tools the courseware development process can be depicted as follows: the author defines a subject for the product to be derived. Then a structure has to be defined, clarified and detailed so that it is clear which material has to be collected and how it has to be organised. This is done building up an empty structure describing the overall architecture skeleton at very high level.

Chosen materials are then collected, digitised, stored and indexed in a suitable manner as to allow an easy retrieval and handling while actually building the final product. Huge database will be stored reflecting the main tree structure of the subject as sketched in the previous step. Items related to a specific presentation will physically be stored with the presentation code itself. According to the planned structure the actual product flow structure has to be sketched. The derived template will have to be filled afterwards when all tree and show components will be ready. The sketched skeleton will be detailed and implemented. This will lead to a detailed empty structure.

When the structure has been prepared it is possible to organise the navigation structure (that is the tree level and the foreseen hypermedia links). Such a navigation structure can be filled when all presentation items will be ready. This means to organise presentations into elementary tree structure which correspond to a lesson and then to organise the single lessons into a module structure. When all preparatory steps have been fulfilled it is possible to prepare the actual set of presentation items. They may consist of still images, texts, sounds, motion pictures, animation. These presentation elements can, then be used to fill in the aforementioned structure templates. This brings to a tree of tree structure with hypermedia links connecting transversally certain tree branches. When all elements have been completed and templates have been filled up, it is possible to start assembling the actual application. Refining the user interface, preparing the help and documentation of the actual product. This phase brings to the validation of both the content and the user interface of the application. In this phase are carried out all necessary steps in order to attain a bug free product.

5. THE TUTOR AND THE STUDENT ENVIRONMENTS

In the present version, the student's fruition of the course is performed in two different ways as shown in the workbench of Fig. 2:

- the stand-alone version allows for using the courseware on a stand-alone multimedia system, being the course distributed on CD-ROM,
- the networked version provides several possibilities of more complex interaction with a tutor for help, guidance, and scoring.

Fruition of the lessons and navigation through them is supported by the same three-layered structure seen for the authoring environment. In this case, only the run-time components for running the lessons are available to the student as embedded tools: no modification to the lessons is supported but only presentation of the multimedia components and navigation through pages and lessons.

Activities of the tutor in course adaptation to specific student's needs or to specific classrooms is supported by the same *Flow layer* used by the author to compose the basic structure of the course. In this environment, the tutor is allowed to modify and merge lessons, created by authors, in new courses; modification of the structure within each lesson is not allowed.

In the networked version, interaction between students and tutors in a LAN or WAN as well as courses distribution are provided by several services.

All system services are available through a common *Service interface layer*. In the picture some services have been pointed out. This is due to the fact that only certain services are of interest while authoring or attending a multimedia courseware. Among these services the most important one is the *Rendering engine*. This module is OS-dependent, but allows access to the OS in an OS-independent manner. Therefore a course prepared on a PC platform using ".avi" and ".wav" files will be executable on an Acorn-Arm machine, to whom such kind of files are unknown, simply because the rendering engine takes charge to present these items to the OS as if they were known ones. The same operation is possible the other way round. Such an approach allows to easily expand cross-portability to Apple and other OS platforms.

Before examining the other relevant services it is necessary to highlight how the overall system will be exploited. It has been previously mentioned that the author will use the runtime environment as a support tool while developing. That means that all services available can be recalled and tested. It is evident, anyhow, that the authoring activity is an off-line one. The fruition activity, on the contrary, is an on-line one. Moreover a courseware can be used either in a stand-alone environment or in a networked one. These are, of course, two totally different environments. The networked one is usually available in structures institutionally providing training services where is also usually available at least a tutor. The working stations are networked and connected to a server.

Users, attending a course, log-on the server and start a curriculum which may have been customised. The tutor can log-on the system, check who is actually logged-on, what activities are being performed by each user (this is done by examining a user related log of active module / lesson). He can also see what are their score and overall performances by checking users' scores recorded in the centralised database. The tutor can also be available for providing help to logged users. In the runtime environment are available a set of tools and services devoted to this purpose (explained later on), but all these specific functions are available only in the *tutor runtime environment* of META system. Usually the end user acquires a fully working basic subset of the runtime environment. Tutor services are an add-on for the basic runtime environment. Such extensions are separately available, but also integrated in the authoring environment add-on. Something similar happens also for the authoring environment, where there is a *basic* version which allows only to operate on the flow level, on the exercises and on the scoring environment. There is then an extension of the authoring environment to bring it to the full-size version. Other relevant services are:

File transfer: allows the author to access remote databases where raw material for courseware development has been stored. It also allows lesson delivery in the distributed environment. By using this service a pupil / professional can down-load the courseware on his system and send back his work for evaluation. It can also allow to down-load off-line help data like FAQs.

E-Mail: allows the tutor and the pupil / professional to keep in touch. It can be used in order to put questions and retrieve related answers. It is the easiest way of contact between teacher / tutor and pupil / professional. It is also the only reasonable way to deal with very large number of students at the same time.

Video telephony: can be used to reinforce tutor support giving the feeling of tutor presence. The service will be available all times a tutor is available at a courseware distribution site in a networked environment. The student can place a *call* to the tutor by pressing a specific button on his interface. The call is queued to the centre. If the tutor is available a connection is established. When this happens the system suspends pupil's normal system interaction, a window, totally devoted to video telephony service, is opened and placed on top of all others and the pupil sees and hears the tutor. The pupil will be able to ask questions and get an answer on the spot. The video-telephony interface window presents an area devoted to video image, one for interface control and an other for written interaction. The written interaction areas allows to bi-directionally see what each subject wants to show to the other in written form. The established link between the two grants transmission bandwidth for video, audio and text at the same time. It is important to notice that the video-telephony service is an application that remains aside of the courseware fruition, when recalled it stops normal courseware fruition as long as it is in use.

Remote screen vision: a tutor can use this service in order to grasp a hard copy of a pupil's station screen. The tutor also receives such a screen hard copy, made at the very moment the pupil has required a video-telephony session. The tutor will be then able to better and sooner figure out which are pupil's doubts and needs. Such service allows a better pupil support as well as a more efficient student control.

Scoring: this service is a very important one, as it provides information both to the user and, eventually, to the tutor. This module take care to keep track of user's performances while attending an exercise session. It stores exercises results, score related information, variables and indexes. When required can provide either a synthetic index of user performances or a full report. Through such service, in the stand-alone environment, the user can have a direct feedback of his progresses and an impartial evaluation of his results / performances.

6. CONCLUSIONS

META project aims and results are described, while project development is presented according to progresses in time. The overall system is presented in terms of components and related functionality through a block diagram. Each system component is described, at least at core level. META derived products development cycle is presented and a few conclusions are drawn on project results.

META project results are expected to cut down production times for multimedia courseware both stand-alone and distributed. Moreover the possibility to assure both a networked and a loosely connected version of the distributed environment represents a major innovation in the training environment. META results are expected to become a *de facto* standard in the training products development environment.

7. REFERENCES

- [1] Functional specifications report, META-ESPRIT Project 8045 deliverable D-1.3
- [2] Report on META system architecture, META-ESPRIT Project 8045 deliverable D-1.4
- [3] Market analysis report, META-ESPRIT Project 8045 deliverable D-1.1
- [4] Market competition report, META-ESPRIT Project 8045 deliverable D-1.2