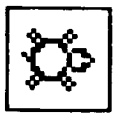
Herramientas de autor para el desarrollo de software educativo

Jose Luis Rodríguez



La gran muralla del ordenador para los educadores ha sido que imponía el juego que Goethe achacaba a los matemáticos: lo traducen todo a su idioma y ya se trata de algo distinto inasequible para la generalidad de los mortales. La informática separaba a los educadores de la informática. Los lenguajes «de autor» que traducen las posibilidades del ordenador a pautas de acción expresadas en códigos sencillos y naturales y que sirven para producir cursos, lecciones o didácticas fácilmente, permiten por primera vez que el educador pase a ser protagonista y no el anfitrión de una visita inoportuna, del papel del ordenador en educación.

En otro lugar (cf. nuestro artículo en este mismo número de *CL&E*) hemos expuesto algunos aspectos del cambio que se produce en el interior de las aplicaciones microinformáticas en entornos educativos con el advenimiento de los interfaces gráficos de usuario y las ventajas que ello comporta. También, con la perspectiva de poder utilizar sistemas de autor de tipo gráfico para la producción de software educativo. Quisiéramos a continuación desarrollar este enfoque, comentando las herramientas disponibles en la actualidad y que pueden ser utilizadas por los profesionales de la educación.

En primer lugar conviene hacer una doble matización: por un lado, aunque hablemos de las herramientas de producción, de programación, no pretendemos obviar la cuestión previa y decisiva del diseño del programa informático/didáctico. Más aún, sin un diseño pedagógicamente adecuado las herramientas solas no sirven para nada. Pero, a la inversa, sin unas herramientas fáciles de usar y potentes el mejor diseño puede quedarse únicamente en el papel. En este trabajo no vamos a hablar de diseño de software educativo (no sólo por una cuestión de espacio, sino por tratarse de un tema más difícil de sistematizar y sobre el que no existe acuerdo entre los investigadores [1].*Ruso)*, sino que nos vamos a concentrar únicamente en las herramientas.

La segunda matización se refiere al porqué desarrollar software educativo. Igual que«el movimiento se demuestra andando», o que el conocimiento se adquiere a través de la acción (y de la reflexión), **el solo uso** de los programas informáticos hechos esplícitamente para fines educativos puede suponer una cierta desmotivación o, por lo menos, una gran distancia por parte del profesor que no puede modificarlos ni adaptarlos a su práctica docente. La idea que subyace a presentar herramientas fáciles de utilizar por profesionales con conocimientos informáticos medios (no expertos programadores), es que puedan plantearse producir, por ellos mismos o, mejor, en el interior de equipos, materiales en soporte informático. Más adelante comentaremos los límites de esta posibilidad.

REFLEXIONES PREVIAS

El proceso de desarrollo de software educativo es más complejo de lo que puede parecer visto desde fuera —a pesar de que se posean herramientas adecuadas—. Las siguientes reflexiones/decisiones podrían ser hechas antes de lanzarse a un proyecto:

a) dicidir si la «implementación» informática del material es necesaria o, como mínimo, conveniente. Dicho de otra forma: a lo mejor la necesidad de elaborar un determinado material didáctico no requiere utilizar ordenadores. No exite, desde luego, un criterio único para tomar esta decisión, aunque algunas reflexiones podrían ser:

- ¿es posible realizar un material didáctico (no informático) semejante de manera fácil? ¿Cubriría las mismas necesidades que el programa informático? Si ambas respuestas son afirmativas quizás es mejor desestimar la aventura (probablemente más costosa en tiempo y en recursos) de realizar sotfware y concentrarse en material didáctico más convencional. Sin embargo, también hay que pensar que, en muchas ocasiones, el cambio de medio supone cambios en la forma de apropiación de los contenidos o de los procedimientos por parte de los sujetos...
- una segunda aproximación es pensar en las ventajas añadidas que supone utilizar el soporte informático: facilidad de repetición, velocidad en el feedback, posible motivación mayor de los alumnos, utilización del ordenador en parejas o grupos, etc. Esto debe ser balanceado con las desventajas consiguientes: la repetición suele ser muy mecánica, el feedback aunque rápido nunca es tan específico como el que proporciona el profesor, determinados sujetos pueden tener una relación nada positiva con la utilización de las máquinas, etc. Como es obvio, sólo el análisis de la tarea, con especial énfasis en los contenidos y en las formas de interacción, puede hacernos pensar en las ventajas diferenciales.

b) decidir si, en una primera aproximación, podemos realizar el desarrollo nosotros mismos, o bien tenemos el equipo personal adecuado para hacerlo.

- hay que pensar si además de ser buenos conocedores del tema que queremos enseñar, seremos capaces de esquematizarlo en términos que sean susceptibles de una implementación informática. Esto quiere decir, que, en el contexto del diseño general del material, hay que prever la necesidad de «traducción» de los contenidos al soporte informático —y que no se trata de un paso fácil.

c) decidir si el material informático que poseemos es adecuado para nuestros fines.

• esta cuestión es relativamente más fácil que las anteriores y se plantea negativamente en relación a los rasgos del programa que se definan por las posibilidades del hardware.

Sin embargo, el material también está condicionado por la herramienta de desarrollo. Las mejores y más fáciles se encuentran para ordenadores que utilizan Windows 3 y para ordenadores Macintosh. Si no poseemos ninguno de los dos, la elección de la herramienta puede condicionar muchos el tipo de software y el proceso de desarrollo que podamos hacer.

Finalmente, la cantidad y calidad del material también condiciona el uso docente que podamos hacer del software, así como el propio desarrollo que, en ocasiones, requiere máquinas más potentes que las que posteriormente se utilizan con alumnos.

d) decidir si queremos o podemos distribuir el material resultante.

• esta no es una decisión que afecta tanto a desarrollar o no el material cuanto a la exigencia de pensarlo para un uso colectivo. La simple trivialidad de colocar como objetivo la posibilidad de que vaya a ser usado por otros conlleva tener que cuidarlo mucho más, explicitar aspectos que para nosotros puedan ser evidentes, y, en general, introduce una cierta evaluación continuada sobre el propio proceso. Sin embargo, no es probablemente una buena opción para los primeros intentos que se hagan, pues el software educativo requiere un grado de finalización muy elevado que no es siempre alcanzable en poco tiempo.

TIPOS DE SOFTWARE EDUCATIVO Y HERRAMIENTAS DE DESARROLLO

Es tradicional distinguir entre varios tipos de software educativo (práctica y ejercitación, tutoriales, simulaciones, etc.), en una clasificación que responde a la vez y de forma poco clara al tipo de conocimientos que se transmiten y a la forma de la interacción que el propio software propone. Además, otros tipos de programas nacidos sin especial intención educativa, sino como de «propósito general» en el campo microinformático (p.e., procesadores de textos, bases de datos, programas de dibujo, hojas de cálculo, juegos, etc.) son también reutilizados con fines educativos. En el caso del desarrollo de software educativo por parte de los propios profesionales educativos, parece más prudente concentrarse sólo en el primer grupo y en algún elemento del segundo —como los juegos.

Dado que el propio medio informático es integrador de otros medios, los programas pueden utilizar sólo las capacidades básicas del ordenador (texto, sonido limitado, gráficos, color, animación, impresión), o bien aumentarlas con otros recursos tales como sonido digitalizado, imágenes externas almacenadas en vídeodisco y otro periférico y control de periféricos en general. Esto conduce a tener que ser muy específicos al referirnos tanto al tipo de software educativo en que pensamos como a las potencialidades de las herramientas.

En relación a este abanico amplio de tipos de software [2] y configuraciones que utilizan nos vamos a concentrar únicamente en aquellos sistemas de desarrollo denominados **de autor** (en adelante **S A**: sistemas de autor, basados en un lenguaje de autor pero aportando más elementos de control del entorno, así como formas definidas para la entrega final del programa realizado). Y dentro de estos a los que utilizan un interfaz de usuario de tipo gráfico [3] .

Empezaremos por una descripción muy somera de las herramientas, seguida de una valoración.

Lenguajes icónicos

Bajo esta expresión nos encontramos con aquellos **S A** que represetan el flujo del programa como un conjunto de iconos enlazados. La representación gráfica **es** el programa mismo [4], de forma que la selección y traslación de un icono a otro lugar del flujo representa un cambio real en el propio programa. Los iconos son objetos o acciones que pueden ser compuestas en el conjunto de una pantalla, y las flechas de unión entre los iconos representan los caminos que un alumno puede seguir —tanto para cambiar como para evaluar una respuesta.

Este tipo de sistemas son completamente visuales, de forma que la estructura del programa se ve en la propia pantalla de edición. Como es lógico, todos ellos soportan una construcción por niveles de forma que un conjunto de iconos (un trozo del programa) puede ser agrupado y colocado jerárquicamente en relación a otros iconos. La mayoría de estos **S A** corren el riesgo de ser demasiado prolijos en su propia forma de representación, dando a veces la sensación de una maraña de relaciones.

Una de las principales características y ventajas de estos sistemas es la facilidad de rectificación de un proyecto. A diferencia de la programación clásica, los **S A** icónicos son una pseudo programación «orientada a objetos», de tal manera que las pantallas completas, partes de las mismas, o secuencias complejas de interacción pueden ser tratadas como un único objeto (un conjunto de iconos agrupados).

Otra característica importante que comparten con los **S A** clásicos es la capacidad de gestionar la interacción del alumno. Todos ellos poseen variables que acumulan los resultados a las preguntas realizadas, o el número de veces y el tiempo empleado en responder una pregunta determinada, etc.

Hay tres **S A** comerciales de tipo icónico que vamos a comentar brevemente: Course-Builer, IconAuthor y Authorware.

• **CourseBuilder.** Se trata del primero o uno de los primeros **S A** icónicos que existieron (las primeras versiones aparecen hacia 1986). Para muchos autores sigue siendo el **S A** más fácil de utilizar de todos los existentes en el mercado y permite un tiempo de aprendizaje muy bajo. Las posibilidades principales que tiene son: editor gráfico incorporado (mapa de pantalla y vectorial), utilización fácil de sonido digitalizado, editor de animaciones incorporado, control de vídeodisco y de otros periféricos (en la version denominada VideoBuilder), gestión del alumno, gestión de informes, control de impresora, control de entrada de datos, puente con otras aplicaciones, variables de usuario definibles, generador de aplicaciones independientes.

Con este abanico de posibilidades incorporadas CourseBuilder permite el diseño de cursos complejos que utilizan toda la panoplia multimedia. Sin embargo, también tiene limitaciones importantes: sólo funciona en entorno Macintosh, no permite escribir «código» sino bajo formas muy elementales, no acepta rutinas externas compiladas. Uniendo las ventajas y desventajas, CourseBuilder es un muy buen ejemplo de **S A** icónico para aquellos autores que empiecen a desarrollar software educativo siempre que utilicen sólo ordenadores Macintosh y que renuncien a programar o a utilizar programas escritos por terceros para incluirlos en sus cursos.

• **IconAuthor.** Este **S A**, también de los más antiguos, es un equivalente aproximado en entorno Windows a **CourseBuilder** —aunque con algunas diferencias importantes.

La versión 3 tiene diversos editores: gráfico, texto, animación, vídeodisco, sonido, variables, etc. Posee, además, un «presentation system» que permite ver y ejecutar las aplicaciones creadas con los editores anteriores. La potencia de IconAuthor está, sin embargo, limitada por el interfaz de desarrollo: a pesar a utilizar una visualización del flujo mediante iconos, los módulos de edición están relativametne separados entre sí, siendo necesario crear primero los gráficos, el texto o la animación en sus editores respectivos para ser luego compuestos en pantalla. IconAuthor integra, como otros productos que funcionan en entorno Windows, una forma de intercomunicación de datos y de rutinas con otras aplicaciones (DDE y DDL).

• **Authorware.** Anteriormente denominado **Course of Action** es, sin duda, el **S A** icónico más interesante de todos los comerciales. Orientado también a la producción de cursos para la formación en empresas, posee unas capacidades superiores a los anteriores: dos tipos de animación incorporada, control de las animaciones por variables, control y gestión del sonido digitalizado, control de vídeodisco utilizable por el usuario, mayor número de variables que los anteriores, creación de modelos reutilizables en otros cursos, pseudo base de datos incorporada, formas de interacción y feedback predifinidas, generación de aplicaciones, utilización de recursos externos en formato XCMD (Macitosh), trasvase inmediato a Windows.

Authorware tiene también sus problemas. En especial no ser tan intuitivo como otros **S A** icónicos. El hecho de que cada icono posea muchas opciones predefinidas, pero con varios valores posibles, hace que el flujo del programa no siempre sea visible de inmediato, y que haya que «abrir» los iconos para entender cómo se van a comportar. Esto requiere un tiempo de aprendizaje superior del que aparentemente es óptimo, así como comprender algunos rasgos de funcionamiento complejos.

El punto crítico de Authorware, sin embargo, es que las aplicaciones creadas en entorno Macintosh funcionan igual en entorno Windows. De esta forma, el trabajo de desarrollo es más aprovechado. Por desgracia, las versiones anteriores de **Course of Action** permitían trasvasar el curso a Ms-DOS estándar (con algunas limitaciones), sin necesidad de Windows, opción que no es soportada en la actualidad.

• Finalmente, hay otros tipos de **S A** icónicos que se están desarrollando en la actualidad y de los que empiezan a existir versiones comerciales. Nos referimos, especialmente, a proyectos de la Comunidad Económica Europea inscritos en el programa Delta. Uno de ellos, ha creado **Orgue,** un **S A** icónico para entornos Ms-DOS.

Orge es un conjunto de programas integrados: un editor gráfico y de textos, un programa para utilizar sonidos digitalizados, un editor lógico del flujo del programa, etc., en conjunto posee características comparables a los **S A** icónicos más avanzados, y los cursos elaborados con Orgue permiten utilizar pantallas de alta resolución en color, pueden controlar un vídeodisco, generan menús en pantalla del tipo encontrado en entornos gráficos, etc. Orgue posee además una posibilidad abierta de escribir rutinas directamente en Pascal que son leídas y compiladas con el resto del programa. De esta forma, los aspectos más comple-

jos del tratamiento de variables pueden ser solucionados de una manera tradicional.

Considerado desde el punto de vista del interfaz de usuario, los diferentes módulos de edición de Orgue dejan todavía mucho que desear. No sólo por el concepto, anticuado pero casi inevitable en este tipo de entorno, de trabajar en distintos módulos y luego integrarlos, sino por el hecho mismo de sólo utilizar la forma compilada en el modo ejecución. Adoptar el punto de vista del usuario final requiere compilar todo el programa, con un proceso de depuración y chequeo de errores incorporado («debugger») en la propia compilación que hace muy laboriosos los procesos de «visionado». En cualquier caso, Orgue es un producto nuevo, que permite utilizar ventajas importantes de los entornos gráficos en Ms-DOS, y del que es posible esperar mejoras importantes.

Lenguajes de guión («scripting»)

Los lenguajes de guión son formas simplificadas de lenguajes de programación que incorporan además muchos elementos de los sistemas de autor. La simplificación consiste en varios aspectos:

- una sintaxis más sencilla, sin apenas paréntesis, comillas, puntos y coma, etc., a lo que se añade una utilización de los verbos para describir acciones mucho más intuitiva, más cercana a los verbos habituales del inglés.
- un manejo muy simplificado de las variables. Sólo deben ser declaradas, local o globalmente, al principio de un script, sin necesidad de hacerlo al inicio del programa, ni tampoco hay que determinar tipos de variables.
- una programación «orientada a objetos», de forma que no es necesario escribir un largo y único programa, sino que cada objeto (botón, campo, página o tarjeta, fondo, pila de tarjetas) posee su propio programa y se comporta de acuerdo con él cuando recibe un mensaje.
- un lenguaje de programación/script interpretado, con búsqueda inmediata de errores sencillos —aunque mucho más lento que los lenguajes de programación tradicionales.

Este conjunto de simplificaciones hace que los lenguajes de script sean, en conjunto, mucho más fáciles de utilizar que los de programación, además de ser más fácilmente modificables. Su potencia, desde luego, es inferior para muchas operaciones, pero más que suficiente para crear programas complejos. Más aún, todos adoptan un interfaz gráfico de usuario que está incorporado en el propio sistema y que puede ser redefinido desde el lenguaje, lo que permite que las aplicaciones creadas tengan un interfaz gráfico con muy poco esfuerzo (creación de botones, gráficos, menús definidos por el usuario, utilización del ratón, etc.).

Otra caracterítica no menos importante de estos **S A** es su carácter abierto. Permiten incorporar código compilado escrito en Pascal o en C para fines específicos que se ejecuta a gran velocidad. De esta forma, sin necesidad de saber programar en estos lenguajes se puede aprovechar la potencia de los mismos mediante instrucciones sencillas ejecutadas desde el propio lenguaje de script. Hay quien ha dicho que, de esta manera, se utiliza lo mejor de ambos mundos.

Los problemas o las limitaciones de los **S A** de script radican, paradójicamente, en su propia potencia. No se trata de «simples» sistemas de autor con una cuantas instrucciones de programación incorporadas, sino que poseen autén-

ticos lenguajes complejos (en el caso de Toolbook, por ejemplo, cuenta con casi 600 instrucciones básicas), más fáciles de utilizar que los de programación pero casi tan compejos como ellos si se pretende hacer un programa amplio y muy detallado. Otra limitación importante radica en su carácter interpretado: suficiente para muchos casos, pero excesivamente lento para aquellos casos en los que se utilizan estructuras de repetición o bucles encastrados con estructuras de decisión en ellos; en estas ocasiones el tiempo de ejecución puede llegar a ser muy alto. Finalmente, el principal problema desde el punto de vista educativo de los **S A** de script es la ausencia de gestión de las respuestas del alumno, así como de los tiempos y repeticiones que se producen; se trata de una limitación relativa, puesto que se puede suplir mediante una programación que declare y recoga los valores de tantas variables como sean necesarias, pero esta solución no deja de estar alejada de la tradición que supone a los **S A** la gestión automática de estos aspectos.

Este tipo de sistemas de autor son, de alguna manera, la superación lógica de las clásicas críticas a los **S A** de los años 70, si bien están más cerca de los lenguajes de programación que de aquellos.

Los principales **S A** de script se parecen mucho entre sí, y la mayoría están adoptando un lenguaje común en un porcentaje muy alto de sus comandos básicos. Esto permite que la programación sea muy parecida en todos ellos (aunque con las peculiaridades de cada uno). A continuación comentaremos los más extendidos:

- Hypercard. Es el primero de todos y el que supuso en el momento de su introducción (1987) un cambio radical en la forma de programar un ordenador con fines educativos. Sólo funciona en ordenadores Macintosh.

Las ventajas de Hypercard pueden pensarse de dos maneras: las referidas al sistema como tal y las derivadas de ser un estándar. Las propias se refieren a las ya comentadas anteriormente para los **S A** de script y a otras más específicas tales como: editor gráfico incorporado muy semejante a MacPaint (el primer programa estándar de dibujo para ordenadores Macintosh) y de fácil utilización, alta velocidad de búsqueda, formato muy compacto de pantallas gráficas, libre completamente de errores, etc. Por otra parte Hypercard es el **S A** más extendido en la actualidad (entre otras razones por formar del software básico entregado con el equipo microinformático, al igual que el sistema operativo) y ha sido, y todavía es, el punto de referencia en torno al cual se diseñan los otros **S A** de script.

En relación a Hypercard se han desarrollado miles de pequeños programas, muchos orientados a la educación, que acrecientan su valor. También existen cientos de rutinas externas que se suman a la capacidad del lenguaje. Por ejemplo, Hypercard no soporta la creación de menús de forma directa pero existe un recurso externo que permite hacerlo fácilmente. Hypercard no permite el control de vídeodiscos de manera propia, pero existen recursos externos que lo hacen fácilmente. La animación gráfica es muy limitada en Hypercard, pero un recurso externo permite «leer» archivos de un programa de animación profesional, **Director**, que pasan a estar integrados en el mismo curso. De esta forma, Hypercard es ante todo una estructura abierta que puede ser rellenada con este tipo de recursos según las necesidades. La potencia de Hypercard sólo puede plantearse en función de estas rutinas. Estos recursos externos tiene un formato definido (denominado XCMD y XFCN, según sean comandos o funciones), que es utilizado por otros programas, como por ejemplo Authorware, o Cuarta

Dimensión; de esta forma, la biblioteca de recursos aumenta con el tiempo al ser utilizada por otro tipo de programas.

Las limitaciones de Hypercard son, sin embargo, muy importantes. Pensado para máquinas de pantalla pequeña y funcionando en blanco y negro, Hypercard sólo accede a pantallas de más de 9 pulgadas a partir de la versión 2.0 (1990), por lo que la mayoría de las aplicaciones anteriores deben ser redimensionadas, y en ocasiones reprogramadas en parte. Limitació mayor sucede con el color: hay recursos que permiten visualizar un gráfico en color, pero no es posible dibujar en color directamente sobre la pantalla. Otro aspecto negativo es que no permite generar aplicaciones independientes: para leer un archivo de Hypercard, el usuario final debe tener el programa en su ordenador, y, a pesar de que existen formas relativamente seguras de proteger los scripts contra modificaciones involuntarias, es difícil garantizar que un curso diseñado no sea modificado en un momento u otro.

Finalmente, la apertura de Hypercard es también un problema cuando se piensa en producir software educativo para distintos entornos informáticos. La traducibilidad de programas Hypercard a otros **S A** de script para entornos Windows (como Plus o Toolbook) se limita sólo a la programación básica, pero no a los recursos externos que no poseen equivalente en Windows. Por tanto, un programa que utilice obligatoriamente un recurso externo no puede ser trasvasado. Los problemas y las posibilidades de trasvase entre distintos entornos puede verse resumida en la Tabla I (volveremos más adelante sobre ello).

- **Plus.** Este **S A** nació con la intención explícita de superar los problemas de diseño de Hypercard. Entre sus ventajas figuran la de utilizar color, tamaño de pantalla variable, así como mejorar los campos mediante una división en campos normales, de procesamiento de textos, y de base de datos (garantizando sólo el tipo de entrada de datos). Otra ventaja importante es la de permitir utilizar una versión «runtime», sólo de lectura, de los documentos creados con Plus, que permite que el usuario final no tenga que poseer Plus.

La principal innovación de Plus es sostener una estructura de datos que permite la traducción inmediata de los documentos creados con Plus en entorno Macintosh a documentos Plus en otros entornos, Windows y OS/2, así como a la inversa. Desde este punto de vista, un desarrollo realizado en Macintosh es inmediatamente utilizable en Windows (¡sin recursos externos, desde luego!), con ligeros cambios en los tipos de letra. Este es evidentemente el camino que parece más adecuado para sobrepasar los problemas de los diferentes estándares de hardware hoy existentes.

Sin embargo, Plus tiene algunas desventajas importantes. Por ejemplo, el no estar completamente libre de errores («bugs») de programación, lo que hace más difícil el desarrollo. Por otra parte, la versión Windows requiere grandes cantidades de memoria para desarrollo, casi nunca presentes en ordenadores escolares (4 ó 5 Mb de memoria RAM). En fin, la versión Windows es extraordinariamente lenta, mucho más lenta que el equivalente en Hypercard, lo que hace que muchas aplicaciones pierdan su interés.

- **Supercard.** También concebido para mejorar los problemas de diseño de Hypercard, este **S A** sólo funciona en entornos Macintosh. No existe posibilidad de traducción alguna de documentos Supercard a otros entornos.

Supercard es, sin duda, el **S A** de script más complejo y potente de todos los existentes en cualquier entorno. Diferenciando un módulo de edición y diseño de un módulo de ejecución, permite al autor controlar mejor los elementos

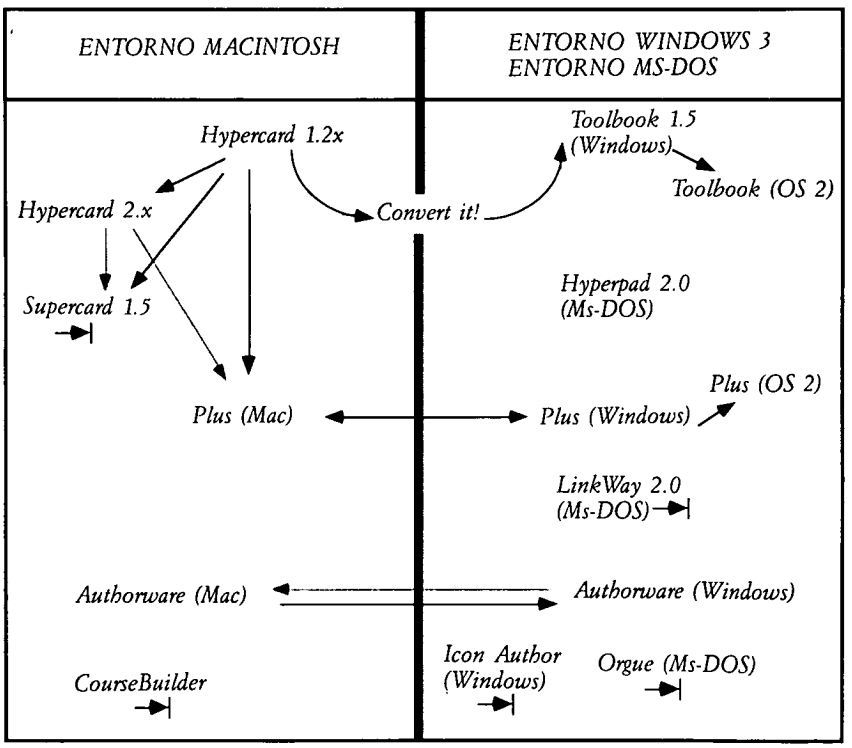
de su programa. Creación automática de menús, multiventana y multidocumento, y generación de aplicaciones independientes, son los rasgos diferenciales más importantes de este **S A.** Las aplicaciones creadas con Supercard son auténticos programas, que pueden generar sus propios documentos.

Las desventajas de Supercard derivan de su complejidad. Es menos intuitivo que los otros **S A**, y el autor debe tener una idea muy clara de lo que desea hacer para afrontar el módulo de edición. Los autores menos experimentados suelen encontrarlo más difícil de utilizar que Hypercard u otros equivalentes.

• **Hyperpad.** Este **S A** funciona en entornos Ms-DOS. Concebido como una réplica a Hypercard, utiliza casi el mismo lenguaje de programación y la misma filosofía de objetos y jerarquía.

Las principales ventajas de Hyperpad derivan de su similitud con Hypercard, adoptando unos principios reconocidos en los sistemas de autor actuales. Utiliza además muy poca memoria, pudiendo funcionar en cualquier ordenador independientemente también de la velocidad de su procesador. Tiene también una versión «runtime» que posibilita la utilización de los documentos

TABLA I



Correspondencias entre los principales sistemas de autor de diferentes entornos y posibilidades de trasvase de uno a otro.

Las flechas negras representan posibilidades actuales de trasvase; las flechas grises posibilidades previstas. El símbolo → se aplica a los casos en los que no existe trasvase posible. El trasvase de documentos y de cursos entre entornos supone conservar las pantallas, con gráficos y color, el texto y la programación. No se conserva el sonido digitalizado, las animaciones exteriores al sistema, ni los recursos externos compilados.

Hyperpad sin necesidad de tener el programa. Otras particularidades son una implementación de una pseudo base de datos, elección de términos de una lista, efectos especiales de paso entre pantallas, soporte del ratón, etc.

Los problemas de Hyperpard para su utilización con fines educativos residen en su incapacidad para el uso de gráficos de manera bien integrada, no poseer editor gráfico alguno, y ausencia de tratamiento de sonido digitalizado. Se trata de un **S A** orientado a la información textual y que permite colocar gráficos elaborados con otros programas como fondo de una pantalla. Hyperpad soporta recursos externos con formato propio, pero no existe una librería de los mismos, de forma que su utilidad real es muy baja. Por otra parte, la versión runtime posee limitaciones muy importantes en relación al programa completo.

- **Toolbook.** Este **S A** se está convirtiendo en el estándard de hecho en el entorno Windows. Pensado también sobre el modelo de Hypercard, con un lenguaje parecido aunque más amplio para soportar los protocolos de intercambio entre aplicaciones que funcionan con Windows 3, Toolbook representa el ejemplo más claro de las posibilidades de un entorno gráfico para ordenadores basados en Ms-DOS/Windows.

Toolbook tiene un editor gráfico, permite la gestión de sonidos digitalizados a través de SounBlaster (una placa para tal fin), posee un lenguaje complejo, permite una animación simple, y, en general, tiene integrados todos los elementos de otros **S A** como Plus o Hypercard. Posee además una versión «runtime» que permite utilizar sus documentos sin el programa original. Sin ser tan lento como Plus, todavía es mejorable en velocidad.

En conjunto, es de esperar que las aplicaciones educativas con Toolbook aumenten mucho en los próximos años, a medida que Windows se convierta en un sistema operativo más usado en ámbitos escolares.

- **LinkWay.** Este **S A** funciona con sistema operativo Ms-DOS con un interfaz de usuario cuasigráfico y con un tipo de programación basada en scripts. A diferencia de los anteriores **S A** comentados, no utiliza el tipo de lenguaje basado en Hypercard sino uno propio más cercano a otros lenguajes de programación (simplificado). Sin embargo, sí que utiliza la metáfora de la composición de pantallas o páginas en términos de botones, campos, páginas, fondos y libros, respetando una jerarquía semejante, aunque no idéntica a los otros **S A.**

Linkway es por tanto un híbrido en relación a los estándares habituales. Sus propias ventajas son muchas: en especial la de permitir utilizar un entorno gráfico en Ms-DOS (a diferencia de Hyperpad), una velocidad de ejecución suficiente, y el tener incorporado un editor gráfico al que «salta» para luego importar los gráficos creados. También permite controlar vídeodiscos, así como disponer de una versión «runtime» para ejecutar los programas creados. El lenguaje de programación que utiliza es un poco más complejo, para autores no iniciados, que el prototípico derivado de Hypercard, aunque permite realizar la mayoría de las funciones que se realizan con este último. También gestiona la utilización de sonidos digitalizados a través de una placa especial.

Las desventajas de LinkWay derivan en parte del propio sistema Ms-DOS y de los distintos estándares que lo soportan. Así, por ejemplo, un programa creado para un tipo de pantalla y resolución debe ser reescrito para que funcione en otra (como ocurre en Hyperpad); los formatos gráficos que soporta son mucho más reducidos que los de otros programas; la integración entre los módulos de editor de textos, editor gráfico, y el propio programa es mucho más deficiente que en otros casos. No soporta animación.

LAS DECISIONES FINALES

Esta visión rápida de los principales **S A** icónicos y de guión que funcionan en entornos gráficos (con dos excepciones cercanas que utilizan metáforas parecidas a pesar de «correr» sobre Ms-Dos) muestra, cuanto menos, la movilidad y cambio de un campo por el que parece existir un gran interés. Sin embargo, no todos ellos están pensados para el sector educativo/de formación. La mayoría de los **S A** de script responden a estrategias actuales y muy generales de la industria informática que, **además**, pueden ser de gran utilidad para los profesionales de la educación. Sólo los **S A** icónicos están concebidos especialmente como superación de los clásicos sistemas de autor —enfatizando, por tanto, la no necesidad de programación—. Como a veces ocurre, una línea de desarrollo industrial muestra sus mejores efectos allí donde sólo va dirigida de manera tangencial.

Ante la situación de empezar a desarrollar software educativo hay varias alternativas claramente jerarquizadas (que también entran en consideración con las mencionadas anteriormente):

(1) diseñar los contenidos y encargar su realización a profesionales informáticos. Esta ha sido la solución más generalmente adoptada en el pasado, dada la barrera tecnológica que suponían los ordenadores. Los resultados, en general, no han sido especialmente satisfactorios por muchas razones —y, a la inversa, muchas de las críticas al software educativo quizás provengan de este predominio de los programadores en el proceso de desarrollo— (sin embargo, debe tenerse en cuenta que todavía en la actualidad proyectos muy complejos requieren la colaboración inexcusable de un programador profesional).

(2) si el profesor se convierte también en autor, y además quiere tener parte activa en el desarrollo, no hay otra solución que «bajar» a la materialidad del proceso —con todos los problemas y desviaciones de la idea original que esto supone—. La decisión se centra en aprender un lenguaje de programación (como Basic, Pascal, Logo, etc.) para lograr que el ordenador ejecute las acciones que se hayan diseñado, o bien aprender un sistema de autor como los ya mencionados. La diferencia principal está en el tiempo necesario para crear materiales didácticos interesantes; tiempo en la propia curva de aprendizaje (mucho más larga para los lenguajes de programación), y tiempo también en la propia realización del programa —también menor utilizando los sistemas de autor—. El único aspecto diferencial son los límites reales de los **S A** para producir materiales muy complejos desde el punto de vista de la interacción instructiva.

(3) Finalmente, si el proyecto de desarrollar software educativo conlleva el aprendizaje y la utilización de un **S A**, conviene pensar qué tipo de lenguaje se va a escoger (de script o icónico), cada uno con sus ventajas y con sus problemas. La Tabla II presenta estas opciones en función de considerar, por una parte, los principales **S A** para entornos gráficos que se utilizan en el mercado informático (sin entrar en otros actualmente en desarrollo, o de uso interno en Universidades, o que requieren entornos informáticos más complejos), y por otra los conocimientos previos del autor.

Este cuadro puede también completarse con el cuadro 1, presentando anteriormente, en el que se resumía las posibilidades de trasvase de los programas hechos en un entorno hacia otros entornos. Como puede verse, la mayor incompatibilidad está entre entornos de interfaz gráfico (Macintosh y Windows) y no

TABLA II

Elección de ***SA*** *en función de los conocimientos previos de programación y del entorno informático.*

Entornos informáticos / **Conocimientos previos e intenciones del autor**	Ms-DOS	Windows	Windows y Macintosh	Macintosh
Sin conocimientos previos de programación y sin intención de aprender «scripting»		IconAuthor Authorware	Authorware	CourseBuilder Authorware
Sin conocimientos previos de programación pero con intención de aprender «scripting»	Hyperpad	Plus Toolbook	Plus	Hypercard Plus
Con conocimientos previos de programación	Orgue Hyperpad Linkway	Toolbook	Hypercad Convert it! Toolbook	Supercard

gráfico (Ms-DOS), mientras que los primeros disponen de herramientas que permiten el paso de uno a otro sin excesivos problemas.

Además de todas estas consideraciones factuales sobre los **S A**, otros dos tipos de reflexiones pueden ser útiles en el momento de plantearse un desarrollo de software educativo con herramientas de autor.

(1) En primer lugar, la cuestión del entorno. Tanto del entorno humano inmediato como del más «lejano», el creado por las propias herramientas.

Lo más habitual es que un proyecto tenga unos antecedentes: que responda a una necesidad de mejorar una didáctica, o que se inscriba en una idea más amplia de cambiar la interacción entre los alumnos mediante tareas compartidas. En cualquier caso, no suele ser el proyecto aislado de un profesor, sino que responde a varios profesores que se implican en el mismo en mayor o menor grado. Es posible que alguno tenga además conocimientos de programación, o conozca a otros posibles programadores: siempre es mejor pedir colaboración en todas las fases del proyecto, pero especialmente cuando se empieza en la programación. Por otra parte, realizar software es una tarea intelectual en sí misma interesante, que todavía lo es más cuando el programa va a ser utilizado en unas condiciones concretas; en ese sentido, el contexto escolar (o de otro tipo) en que un desarrollo va a inscribirse debe ser muy valorado en toda su complejidad.

En segundo contexto es el que determina las ayudas técnicas que provienen del propio **S A**. Algunos de los mencionados responden más a una cultura de formación en la empresa que a una cultura educativa/escolar. La diferencia estriba en que resulta mucho más difícil obtener ejemplos ya hechos, así como intercambios con otros autores/programadores. Por el contrario, aquellos que han nacido con una filosofía «abierta» no suelen proteger sus scripts, o simplemente

disponen de una amplia gama de ejemplos en concepto de «freeware» o de «shareware». Otra variante del contexto de la herramienta es la de poder disponer de bibliotecas de recursos compilados («shareware» o de muy bajo costo), lo que posibilita ampliar las potencialidades del programa.

(2) En segundo lugar, está la reflexión propiamente pedagógica. En parte, es una especificación del contexto de aplicación al que nos hemos referido, desde el punto de vista de la inserción «curricular» del software. Conviene pensar cómo se va a aplicar el programa, con qué tipo de dinámica (p.e., si los alumnos van a esta solos con el ordenador, si el programa sólo va a servir de ejemplificación de aspectos concretos de una explicación verbal más amplia, si va a ser autosuficiente pero requiere el apoyo-guía por parte del profesor, etc.), a qué alumnos va dirigido, a qué curso, nivel o materia, etc. Estas consideraciones son, en el caso de la enseñanza formal, tan importantes como el propio programa —si no más.

Por otra parte, el control que se tiene sobre la secuencia instructiva (autoincluida) generada mediante el programa no deja de ser muy problemático. De alguna forma la «encapsulación» instructiva que se produce en el material informático es doble: la más explícita se refiere a las secuencias instructivas en las que se controla la forma de la interacción, regulando las formas en que el alumno interactúa con el programa, así como los tipo de refuerzo y análisis de la respuesta que se pueden prever. Se trata del diseño instructivo propiamente dicho que los **S A** permiten controlar de manera eficaz dentro de sus límites (casi no incluyendo, por ejemplo, capacidades «inteligentes» en el análisis de la respuesta[5]).

La otra, más implícita, es el propio diseño del interfaz, que aunque no sea independiente de las formas de interacción instructiva, tiene su propio espacio y sus propias leyes que también determinan las formas de aprendizaje[6]. Ambas cuestiones están mediatizadas por los **S A** que las posibilitan limitándolas. En este sentido, nunca debe olvidarse que el mejor **S A** es sólo una herramienta y que su valor depende de quién y cómo la use.

Notas

[1] *El «diseño instructivo» es una expresión que se utiliza para reunir en una disciplina el conjunto de decisiones didácticas sobre el tipo y formas de presentación de unos determinados materiales, en función de unos objetivos y de un grupo determinados. La aplicación del diseño instructivo a la creación de «cursos» informáticos es una (sub)disciplina que tiende a plantear sus propios problemas —en ocasiones muy complejos. Una muestra representativa de las posiciones norteamericanas sobre el tema, en Jonassen (ed, 1988).*

[2] *Nos hemos referido hasta el momentos a programas cerrados o semicerrados, prototípicos de la EAO. El caso de los programas abiertos, mucho más ideal, es distinto. Igual sucede con entornos de simulación «abiertos» —en los que el profesor o los propios alumnos pueden variar las condiciones y componentes de la simulación—. Mención de excepción también la constituyen el caso de los sistemas organizados «hipertextualmente», en los que muchas características del diseño instructivo «encapsulado» dependen de los caminos de interacción elegidos por el usuario. Estas tres excepciones (EAO abierta, simulaciones e hipertextos) son cualitativamente distintas, a pesar de ser todavía cuantitativamente poco significativas.*

En el ICE de la Universidad de Barcelona se están desarrollando dos proyectos para utilizar este tipo de programas en situaciones concretas de enseñanza: uno consiste en la experimentación con un paquete de simulación abierto, Interactive Physics, en la enseñanza de 2.º de BUP y COU de partes del programa de Física; el otro, en el desarrollo de un sistema hipertextual para enseñar concepciones sobre el aprendizaje y sus relaciones con la educación a estudiantes universitarios de Pedagogía y Psicología.

[3] Para un análisis comparativo de sistemas de autor que no utilizan interfaces de usuario gráficos (que funcionan casi exclusivamente bajo Ms-DOS), el estudio del Centro Nacional de Vídeo Interactivo británico (Starwford, 1988) sigue siendo un buen punto de referencia. Este tipo de sistemas va cayendo cada vez más en desuso por su mayor complejidad, tanto desde el punto de vista del interfaz de usuario para el propio autor como incluso por los recursos técnicos que incorporan.

[4] En realidad es una representación del programa, del código. Pero una representación que mantiene una correspondencia estricta.

[5] La posibilidad de utilizar programas «inteligentes» aplicados a la educación está todavía lejos de ser una realidad habitual. No sólo por la falta de desarrollos concretos, sino por la propia complejidad y dificultades del campo de la inteligencia Artificial aplicado a la realización de «tutores inteligentes» que se encuentra sumergido todavía en un proceso de clarificación tanto de objetivos como de las propias herramientas. Una revisión del campo, así como del enfoque alternativo de utilizar programas expertos para *diseñar* software educativo, en Gros y Rodríguez Illera (1991).

[6] La cuestión del diseño del interfaz excede las decisiones básicas sobre desarrollar o no software educativo y con qué herramienta. Sin embargo, aparecen siempre en cada caso concreto por ser inevitables. Shneiderman (1987) es quizás el punto de referencia que ofrece una visión de conjunto más sistemática sobre el tema.

Referencias

JONASSEN, D. H. (1988): *Instructional Desing for Microcomputer Courseware.* Hillsdale, Lawrence Erlbaum.

GROS, B. Y RODRÍGUEZ ILLERA, J. L. (1991): «Inteligencia Artificial y Diseño de programas educativos», *Revista Española de Pedagogía, 88,* 39-57.

SHNEIDERMAN, B. (1987): *Designing the User Interface.* Reading, Addison-Wesley.

STRAWFORD, G. (1988): *Authoring Packages. A Comparative Report.* London, Nacional Interactive Video Centre.

Herramientas de autor para el desarrollo de software educativo
José Luis Rodríguez
CL&E, 1992, 13, pp. 111-124

Resumen: En este artículo se revisan las principales herramientas de autor (sistemas y lenguajes de autor) que están disponibles hoy día par microordenadores con interfaz gráfico, señalando las ventajas que aportan para producir software educativo. Se distingue entre lenguajes de autor de tipo icónico y lenguajes de autor de «script» o guión, analizándolos comparativamente. La descripción está realizada pensando en profesores con pocos conocimientos de informática pero con la intención de elaborar software en el interior de algún equipo.

Datos sobre el autor: José Luis Rodríguez Illera trabaja en el Instituto de Ciencias de la Educación, División de Ciencias de la Educación, de la Universidad de Barcelona.

Dirección: Universidad de Barcelona. Carrer dels Angels, 18. 08001 Barcelona.