Issues in making courseware exploitable

and issues in making exploitable courseware

Douglas Siviter* and Phil Siviter**

*School of Computing, Information Systems and Mathematics, South Bank University

**Department of Computing, University of Brighton

Part 1 of the paper, 'Issues in making courseware exploitable', is about dealing with the legacy of large volumes of incompatible non-integrated courseware which are currently being generated within initiatives such as the Teaching and Learning Technology Programme (TLTP). We suggest strategies for allowing end-users to apply courseware management techniques belatedly to current courseware developments, thereby offering ways of making the emerging courseware more exploitable than it otherwise would be. Part 2 of the paper, 'Issues in making exploitable courseware', takes a forward-looking approach which recognizes that future courseware development efforts must pre-empt these problems of incompatibility and non-integration. Courseware development must mature to the stage where it makes use of courseware design standards, embraces a host of essential lessons from conventional software development, and recognizes the importance of courseware management issues.

Introduction

There are currently substantial investments being made in courseware development in the UK, in particular in the HEFCs' Teaching and Learning Technology Programme (TLTP), and the hope which underpins these development efforts is that Higher Education institutions will eagerly adopt and exploit the freely available courseware which emerges from them. Unfortunately, there are many reasons why this is probably going to be an over-optimistic assumption (Laurillard *et al.*, 1993). The bottlenecks which can impair the successful exploitation of courseware range from cultural to technical, and are quite diverse. In this paper we discuss some of these bottleneck issues and suggest approaches to dealing with them. We deliberately restrict our discussion to technical bottlenecks, and therefore discuss issues of courseware design, courseware development, and courseware management

We first suggest strategies for dealing with a current courseware management challenge, i.e. is it possible to take the current legacy of large volumes of incompatible non-integrated courseware and somehow make it feel more integrated and compatible than it actually is, thereby enhancing its exploitability?

Sorting out courseware management 'after the event' is of course the wrong way to proceed (but inevitable if the currently emerging courseware is to be exploited at all). We therefore take a forward-looking approach and suggest that it is essential and inevitable for the courseware-development community to move forwards to a technical culture which embraces courseware design standards (Jacobs, 1993) and embraces a host of essential lessons from the mainstream software development world. In such a culture, courseware-management facilities will not be a totally neglected aspect of development as they are today, but will instead be regarded as an essential aspect of the courseware life-cycle. Courseware management will also benefit from dealing with courseware objects which are substantially more co-operative than today's 'dumb' courseware.

Part I Issues in making courseware exploitable (tidying up the current mess)

Large-scale vs. Small-scale, Courseware vs. Lessonware

Within the UK there are now many courseware-development efforts which have progressed well beyond the small cottage-industry approach, and there is a growing trend towards large-scale developments via consortia of institutions. Collaborative developments are increasingly being shown to be more viable (both educationally and economically) than small-scale isolated developments. It is not just the scale of the development activities which has substantially increased; the courseware itself is larger and more sophisticated than it has ever been (a fact which is mostly attributable to advances in computer software and hardware). Most previous developments of computer-assisted learning material would be more appropriately labelled as *lessonware*; they usually tackle only a small fraction of what anyone in an academic institution would call a course. By contrast with previous small-scale lessonware products, today's consortium-based developments are trying to produce large-scale courseware which might embody hundreds of pieces of lessonware. As the size and sophistication of the courseware grows, and as more complex patterns of use emerge, many new problems are encountered and new courseware management requirements emerge.

Limitations of current authoring tools

It is, of course, tempting simply to extrapolate upwards and assume that large-scale courseware can be built out of lots of pieces of small-scale lessonware, an assumption which is fine in principle but complicated in practice. Though there are now many wonderful software packages which support the development of small-scale lessonware, there is much less software support for managing the development and delivery at the larger courseware level. Authors can (and do) argue endlessly over their favourite development tools, but they are usually just comparing the built-in facilities for making clever things happen on screen. None of these lessonware tools provide ready-made sophisticated support for developing and endlessly reconfiguring very large collections of CAL modules; nor do they offer sophisticated support for integrating CAL resources into Computer Supported Collaborative Learning frameworks; nor do they offer sophisticated support for integrating resources developed using a diverse range of software tools and applications. These excellent lessonware tools are simply not geared up to address large-scale courseware management issues.

Problems of diversity

Most of the consortium-based developments in the UK at this time have realized that off-the-shelf solutions to courseware management requirements are not readily available, and hence they have either resorted to building their own solutions or have sadly neglected the problems altogether. There are now far too many instances of projects reinventing each other's wheels, i.e. they individually discover the same sets of problems, then partly solve them with in-house developments while remaining totally oblivious to anyone else solving the same problems. Usually, the in-house solutions have failed to consider the extra complexities of ensuring that their products remain open, and almost no effort has gone into pursuing compatibility with solutions produced by different projects. Far from removing technical bottlenecks, we now have a culture of generating them.

As more sophisticated courseware becomes more widely available, academic staff will try to experiment with new ways of exploiting it (we hope). We should recognize that if willing and eager university staff (those not suffering too badly from the Not-invented-here syndrome) try to exploit the currently emerging courseware, they will face a range of technical barriers which will probably deter their enthusiasm and make them give up. The current courseware development efforts are producing diverse courseware. Diversity in an educational sense (in content and in pedagogic style) is an asset, but accidental diversity in courseware management techniques and in basic software management techniques just translates to bewildering incompatibility for the typical end-user, and is therefore a liability.

Partial solutions to current courseware management problems

The courseware development scene is still very young and has not yet matured to the stage where it collectively embraces useful, non-restrictive standards for design and development. As a result of this, end-users now face large volumes of incompatible non-integrated courseware from a diverse range of courseware development initiatives.

We present a framework called HyperCourseware which immediately offers partial solutions to these problems. In the latter part of this paper, we also assert that the HyperCourseware framework can be expanded to form the basis for genuinely open courseware which can achieve substantially higher levels of courseware compatibility.

The HyperCourseware framework

HyperCourseware is not a new idea; our first attempts at providing a HyperCourseware Management System date back to a project which started in 1989 (Siviter & Brown, 1992). A minimal description of HyperCourseware structuring principles is as follows:

- A Course is a hierarchical network of Topics.
- A Topic is a collection of Educational Activities.
- An Educational Activity is . . . (*virtually anything that an author decides*):
  - — described educationally e.g. presentations, assignments, assessments, simulations;
  - — technically implemented using anything, e.g. authoring tools, programming languages, software applications (spreadsheets, databases, etc.), multimedia audio-visual applications, away-from-machine activities.

The above framework is sufficiently general to allow virtually any courseware structure to be generated, but sufficiently precise to allow software tools to support the manipulation of such structures. A HyperCourseware Management System is a set of software tools which enables the creation and endless adaptation of courseware structures and which partially or totally automates the creation of interactive views of the courseware structures. A HyperCourseware Management System therefore enables non-programmers to manipulate large collections of 'lessonware objects' which may have been developed using a diverse range of authoring tools.

Dealing with the legacy of incompatible courseware

A classic scenario within universities is for lecturers to pick and mix from a variety of texts in order to deliver the particular perspective on a subject which they feel is most appropriate. It is a rare occurrence for any lecturer to run a unit totally based on a single set textbook. For courseware to be acceptable to academic staff, it must offer the same kind of pick and mix adaptability that text sources currently provide. Academic staff must have realistic options locally to customize courseware or even embark on major reconfigurations of the material – the courseware must be capable of evolving. As the simplest of examples, a lecturer might like to edit the Aims and Objectives section of a piece of courseware the day before she uses it with a new group of students. A more involved example might be a lecturer creating a totally new courseware structure but populating it with many pieces of existing lessonware, i.e. no new development of resources takes place but a new course is created out of existing resources. Is this courseware development or courseware management?

There is now a national requirement for efforts to be made to bring together the many strands of courseware development. Consumer universities (i.e. all of us) need to have much higher level courseware-management facilities which shield them from technical incompatibilities within the courseware. In effect, there needs to be an after-the-event fudge towards integration which is now a necessary but a poor substitute for integrated design. HyperCourseware can make some contribution to this belated integration. It offers conceptual support by providing a formalized but non-restrictive framework for courseware structures. It offers practical support in the form of HyperCourseware Management Systems: software tools and templates which directly support end-users in courseware manipulation.

Part 2 Issues in making exploitable courseware (avoiding a future mess)

The previous section discussed after-the-event fudges which are now required in order to allow existing diverse lessonware to be used and re-used in larger courseware structures and hence be more exploitable. This after-the-event fix can at best be described as a shallow integration which relies on a fairly crude management of multiple software applications and the generation of multiple 'educationally friendly' views of available resources. How can this shallow integration be improved upon in future? What can we learn from many years of producing incompatible courseware?

The need to learn from the software development world

Within this discussion, it is useful to regard courseware as just a specialized form of software. Obviously, educationists can point out an infinite number of ways in which courseware differs from other forms of software, but there is no escaping the fact that courseware is still software, and there are many lessons which courseware development could and should learn from other areas of software development. The world of commercial software has been forced to address issues of software management, software development, software evolution and maintenance, interoperability of software, etc. The courseware-development community in comparison seems to be barely aware of the problems looming over the horizon. This is tragic, given that an awareness of these problems might have enabled the courseware-development projects to exploit some of the solutions which have emerged within other software development scenarios.

Courseware standards vs. software standards

There will always be a requirement for unrestrained, exploratory development, especially in areas as sophisticated as education, but there are also immense benefits to be derived from standardization in areas which are frankly boring, routine software-development issues. Nobody in their right mind would look at a PC in 1994 and say: 'Because I don't like Windows I would prefer to build my own graphical user interface'. Yet similar nonsensical decisions have been made in relative ignorance about a range of software development issues. It is salutary to note that the art of reinventing boring old-fashioned wheels is alive and well within the world of courseware development (and, as many observers have pointed out, because the courseware development world is relatively new and inexperienced, the reinvented wheels frequently turn out to be square).

The time is right (indeed, overdue) for a much greater effort, preferably on a national scale, to be directed towards establishing courseware development standards. Before, the instructional design experts scream about how prematurely suffocating these standards might be, we would emphasize that the areas best suited for standardization are not concerned directly with instructional design but with the underlying software development issues. The standardization efforts we are advocating have a clear aim of enabling exploratory instructional design rather than constraining it.

It must be appropriate for the courseware development community to recognize that it is wasting opportunities to benefit from current experiences from mainstream software development. Furthermore, given that mainstream software development is advancing more quickly than courseware development, it would be preferable specifically to try to close the experience gap by looking forward to emerging software trends. There are many examples, e.g. there are massive increases in software productivity to be gained from exploiting reusable object-based software components. Microsoft's Object Linking and Embedding, Apple's OpenDoc, and IBM's Distributed System Object Model are examples of a software-development idea which will have a profound impact on how all software (and therefore also courseware) will be developed. Yet it is probably true that only a small minority of courseware developers have any idea of what these ideas represent, and an even smaller proportion are gearing up ready to exploit these inevitable ideas.

There are two overlapping areas of concern. One is as mentioned above: what can (or must) courseware development learn from software development? The other is the identification of generic issues purely within courseware development (generic issues which may have only tentative parallels in software development) and the attempt to engender standards for dealing with these generic courseware development issues.

Moving HyperCourseware towards a Framework for Open Courseware

One on-going development within our HyperCourseware projects has been to map out the courseware development terrain in a way which can draw attention to areas which all courseware development projects have in common, i.e. to identify what are genuinely generic problems associated with courseware management, courseware design, courseware development, and so on. The on-going development of HyperCourseware has also been concerned with providing techniques and tools for addressing these generic issues. The HyperCourseware framework strives to be totally flexible so that in principle any approach to courseware development can be mapped onto the framework, somehow. As a result of this work we now have:

- a conceptual framework for courseware and a vocabulary which can be used a reference model, i.e. any courseware design can be compared with the reference model, if only to verify that the design has, in some way, suitably addressed a known collection of essential issues;
- demonstrator products which practically illustrate many courseware design issues;
- a collection of tools and templates which genuinely enable courseware developers to avoid being distracted by the mundane reinvention of software-development wheels, and instead to raise their sights to the genuinely difficult aims of good instructional design.

Current developments

Our HyperCourseware Management System is object-based. It has well-defined internal protocols which allow us endlessly to evolve the tools and which allow end-users to evolve courseware. Even so, it is fair to say that HyperCourseware is not yet a truly open system. It currently provides an elegant way of linking diverse applications (as do a few other systems such as Microcosm from Southampton University), but these approaches are still well short of true openness. Much of our current research effort is going towards substantially upgrading the HyperCourseware Reference Model so that it can seriously contribute to a Framework for Open Courseware. In effect, we are now developing distributed HyperCourseware.

The software-development world pursues goals like 'exceptional ease of use', 'application integration', and 'evolutionary development'. These are all goals which the courseware development world also needs to pursue. Some of the most dramatic gains in software development are emerging from strategies which apply object-orientation techniques to client/server systems. By adopting similar strategies for courseware development it should be possible to establish a Framework for Open Courseware based on the idea of Co-operative Courseware Objects.

Co-operative Courseware Objects are of course communicating objects which require standard protocols. We believe that the internal protocols for our current Hyper-Courseware system can form the basis for such protocols, hence we are currently engaged in a redesign and redevelopment effort in order to demonstrate this.

We also believe that a Framework for Open Courseware is an eventual necessity for the courseware development community. We are tentatively trying to explore what it might look like.

References

Jacobs, G. (1993), 'Standards', *Association for Learning Technology Journal*, **1**, 1, 2–3.

Laurillard, D. Swift, B. & Darby, J. (1993), 'Academics' use of courseware materials: a survey', *Association for Learning Technology Journal*, **1**, 1, 4–14.

Siviter, D. & Brown, K. (1992), 'HyperCourseware', *Computers and Education*, **18**, 1–3, January, 163–70.