Chapter 7: Reusable Educational Software: a basis for generic learning activities

Diana Laurillard and Patrick McAndrew

Commentary by Peter Sloep

Educational Expertise Centre
Open University of the Netherlands
and
Fontys University of Professional Education
Peter.Sloep@ou.nl
P.Sloep@fontys.nl

Summary

The authors of Chapter 7 advocate the use of reusable software. They adduce a number of reasons, the most important one being that reusable software provides a perfect foundation for generic learning activities. In this commentary I do not challenge the basic soundness of their arguments. Rather, I try to argue that their approach should be taken one step further, from generic learning activities to generic educational designs, and from design specific software to design agnostic software.

Introduction

Laurillard and McAndrew have written a very enlightening chapter on the reuse of learning materials. It is set against the backdrop of Laurillard's well-known Conversational Framework [1]. Fundamental to the argument is that our current teaching practices must address two challenges. First, technological innovations deeply influence our present society - education and training not exempted. They offer all kinds of opportunities for teaching. But to seize them, our academics – and I presume teachers in general - have to possess an innovative attitude and certain technological skills. Second, the traditional transmission model of teaching that we have practiced for so many years, no longer answers to the demands of our knowledge-based society (But see [2]). We need more sophisticated models. And, of course, technological innovation, it is argued, should help us in building these models. (See also [3].)

A problem is that - for lack of training, time, or interest - our academics-teachers are ill equipped to participate in this transition, let alone lead it. However, leaving it to support staff and publishers would be a recipe for disaster: "what we teach is inextricably embedded in how we teach", Laurillard and McAndrew quite correctly argue. So how do we make sure our teaching staff is up to the daunting task of implementing the new pedagogies, whilst making use of the new technologies and, not unimportantly, avoiding an increase in spending on education?

Laurillard and McAndrew offer a solution, at the core of which lie what they refer to as 'generic learning activities'. These learning activities are construed according to Laurillard's Conversational Framework and subsequently embedded in software. The Conversational Framework ensures that modern, constructivist educational conceptions find their way into the learning activities; using software as their matrix guarantees access to modern learning technologies, for instance multimedia

capabilities, but also Internet driven facilities such as web resources and groupware. Finally, by making the learning activities generic, the activities have the quality of templates that can be reused. Not only does this cut costs, it also allows the less technically adroit teacher to create both technically sophisticated and educationally innovative learning activities.

Comments

Although one may question the feasibility of their approach, this is not a line of argument I would like to pursue. I believe their approach to be basically sound. A focus on interactive *activities* rather than content *objects* makes perfect sense, as it is through dialogue - if only internal dialogue - that we learn. And similarly, using software as a carrier for technological innovation seems very plausible in this day and age. So far, so good. But building software demands expertise that the average teacher does not possess. Nor is it feasible to train teachers to acquire this expertise, in addition to the subject matter expertise and pedagogical expertise they also need. Customisable software, easily customisable software to be more specific, would seem to meet the needs of modern, innovative teaching and the capabilities of the modern teacher half way.

The authors discuss a number of examples of this approach. I myself have pursued this avenue, when attempting to create templates for authentic learning in a simulated, web-based enterprise [4]. The flaws associated with the approach reported by Laurillard and McAndrew sound familiar: the adapted product invariably seems to be of a lesser pedagogical quality than the original from which the template originated (or, as was my admittedly anecdotal experience, one spends so much time on avoiding these that the benefits of a template approach become vanishingly small). To this, I would want to add another problem. The software template, generic though it may be, is also specific in the sense that it harbours one specific pedagogical approach only. This means that new software needs to be developed for each didactic variant. What if it were possible to build software that would be able to support not only various instantiations of a particularly pedagogy but also a variety of different pedagogies? That would mean a dramatic extension to the approach advocated by Laurillard and McAndrew. It would in principle also lower the 'unit cost' even further. The crucial question of course is whether this can be done without significantly degrading quality. I believe it can, and in the next section I offer a sketchy explanation of how it may be accomplished. I'll call this section ‘generic pedagogies’ to indicate my intention to expand the authors' notion of generic learning activities.

Generic pedagogies

Software is written in powerful yet uninviting programming languages. Endowing software solutions with a modicum of flexibility therefore cannot be achieved by giving users access to the programming language itself. The software itself must generate in its user-interface some affordances, some ‘dials and knobs’ that one may turn in order to change the user experience. In this particular case, it is the teacher who does the turning and the student whose experience is altered. Although this works, the software developer sets the limits of change. As a software designer develops the software to suit a particular educational design, turning the knobs and

dials may result in a different suite of learning activities, but hardly in a different pedagogy or didactic scenario.

The trick is to not take a particular design as a starting point for generalisation. One may accomplish this by developing a generic language with which any educational design - along with the collection of learning activities modelled according to it - can be described. This language is *generic* in the sense that it covers *all* pedagogies, but *specific* in the sense that it covers pedagogies *only* (and not, say, computer games). So it is much less powerful than a full-blown programming language, but still quite powerful in the context of learning. Having such a language, one still needs to develop software that can 'understand' it and render the educational events captured by it through a user interface. Importantly, one piece of software in principle suffices to support many different pedagogies. Any collection of learning (and support) activities, modelled according to any pedagogy or learning design can be accessed by students and teachers. This language actually exists. It is the IMS Learning Design Specification, which was made public in early 2003 [5]. Players (software applications) for it do not exist yet, although various implementation projects are underway. So we have a technological innovation that supports the much-needed educational innovation, apparently at a significantly decreased unit cost.

Qualms

But what about the ease of use that Laurillard and McAndrew rightly stress so much in their chapter? And what about the quality of the experience offered. On both accounts, the verdict is still out.

Although Learning Design's modelling language is significantly simpler than a full-blown programming language, it is still too hard for most teachers. This may be remedied by creating another piece of software, a Learning Design editor. The situation may be compared to web editing in html. In the early days, html was hand-coded in generic text editors. Subsequently, various generations of dedicated html editors were developed and now everybody can put up a decent website (although for really powerful applications a text editor still is an indispensable tool). Something similar should happen with respect to Learning Design. Currently, only generic text editors exist. Ultimately, it is to be hoped that dedicated LD editors will be built that are powerful yet sufficiently simple to be used extensively by teachers. Since LD is a public and open specification, perhaps a range of editors will be developed, from simple ones that are geared towards one particular learning design each, to complex ones that are template driven and capable of addressing a whole raft of different designs.

As already mentioned, no software capable of playing Learning Design exists at present. However, there is some experience with software capable of running EML, the educational modelling language developed by the Open University of the Netherlands [6] after which the Learning Design specification was modelled. This experience does not suggest that being exposed to an almost identical user interface across different pedagogies decisively influences the student's perception of quality. I'm careful in my wording here, quite on purpose, as little to no systematic research has been published on this subject. What we do know, of course, is that the design of the user interface itself is a decisive factor; but that is a different matter. [7]

Conclusion

In conclusion, the generic learning activity approach sketched by Laurillard and McAndrew is very much in line with the ideas behind the IMS Learning Design Specification, it would seem. Even better, although I say this with some hesitation for lack of rigorous empirical evidence, the goals Laurillard and McAndrew attempt to achieve very likely stand to profit from a Learning Design implementation. I would suggest transforming the notion of a generic learning activity into that of an LD-template. Such a template represents a particular didactic approach or scenario and may be filled with content – be instantiated – thus resulting in a suit of concrete learning activities. We may dispense with design specific software and rely on generic LD players.

References

[1] Laurillard, D. (2003) *Rethinking University Teaching: A Conversational Framework for the Effective Use of Learning Technologies* (2nd ed.) London, Routledge Falmer.

[2] Dai Griffith and Rocio Garcia in their commentary on Koper's Chapter 5 (Combining re-Usable Learning Resources to Pedagogical Purposeful Units of Learning) point to the pervasivess of a conduit metaphor in our language on knowledge acquisition and exchange. This suggest, quite intriguingly, that perhaps we never *actually* taught according to a transmission model, but by the metaphor were misguided in thinking that we actually did.

[3] Sloep, P.B. (submitted) Learning Objects: the Answer to the Knowledge Economy's Predicament?

[4] Westera, W., P.B.Sloep, J.Gerrissen (2000) The Design of the Virtual Company; Synergism of Learning and Working in a Networked Environment. *Innovations in Education and Training International* **37** (1), pp. 24-33.

[5] IMS Learning Design Specification [http://www.imsglobal.org/learningdesign/ The language I'm referring to really is the xml-binding of the specification's information model.

[6] see http://eml.ou.nl

[7] Learning design also possess a plug-in capability. The specification contains a services element that allows one to hook up, say, a groupware product to the LD player which via the services element will be seeded with runtime information. Via the property mechanism information may also be fed back to the LD player. This way, the user experience may be enriched significantly, whilst, from a design perspective, maintaining the integrity of runtime environment. See the LD specification [5] for more details.