Pre-publication draft of a paper which was published in the Journal of Computer Assisted Learning (1989) **5**, 103-113, Blackwell Publishing

Courseware abstraction: reducing development costs while producing qualitative improvements in CAL

G. I. Webb
Division of Computing and Mathematics, Deakin University

Abstract

Courseware abstraction is an approach to CAL whereby the lesson author creates a general parametetized CAL lesson that is then applied to many concrete examples. This approach has the following advantages over alternative approaches to lesson development: it is cost efficient; it facilitates lesson verification; it encourages the provision of as many examples as are desirable; it simplifies the selection of appropriate examples for presentation to each student; it provides a convenient framework for student evaluation, and it supports the development of factually exhaustive lessons. In short it provides qualitative improvements, while at the same time reducing lesson development costs. Although widely used, courseware abstraction has not previously been identified as an important CAL technique and its relative merits have never received attention. In particular, there has been a failure to recognize that generative CAL derives most of its power from the use of courseware abstraction.

Keywords:

Courseware development, Generative CAL.

1 Introduction

This paper describes an approach to the development and presentation of computer assisted learning (CAL) material that provides highly efficient lesson authoring, while aiding the development of sound lessons. This CAL methodology is called '*courseware abstraction.*'

Courseware abstraction is the creation of a single body of CAL material that can be applied to multiple examples. This provides the student with multiple CAL interactions, examining a series of concrete examples within the context of the one general and consistent treatment. Many benefits flow from uncoupling courseware from the specific examples that are examined. For a start, the use of courseware abstraction results in very low authoring costs in proportion to the amount of lesson produced, when large numbers of examples are provided. Furthermore, the benefits of course abstraction do not end with the facilitation of low authoring overheads. It will be shown that the use of courseware abstraction encourages the creation of more detailed lessons than alternative approaches. It also aids lesson verification. Finally, it greatly facilitates student evaluation and the development of lessons that adapt to students' individual educational requirements.

Courseware abstraction is already in wide use in the CAL community. However, it has not previously been recognized as an important C technique. There has been no recognition of the many benefits that it has to offer. This paper seeks to rectify that situation.

2 Lessons that examine multiple examples

It is often desirable to create lessons in which the student is given the opportunity to examine many examples or problems (hereafter collectively referred to as 'examples') from a general body of related examples. For instance, in most disciplines when a new principle is introduced it is usually highly beneficial for the student to be given practice in its application by providing a number of relevant exercises.

The most useful manner in which to provide such practice is to create a large series of exercises, $E_1, E_2, \ldots E_n$ each of which examines a separate example, X_i, such that for all indices, i, E_i examines X_i. The most straightforward approach to the authoring of such a lesson is to create a list of the examples, $X_1, X_2, \ldots X_n$ to be examined and for each of these examples, X_i write a CAL segment, C_i such that C_i generates the interactions with the student for an exercise, E_i examining X_i. That is, using the lesson author's preferred CAL style, a separate segment of CAL code is created for every exercise that the student is to receive. This will be called the *'direct approach'*. It can be seen that, for the direct approach, the provision of *n*, exercises will require the creation of *n* CAL segments.

In contrast, courseware abstraction requires the creation of one CAL segment, *C** and descriptions, $D_1, D_2, \ldots D_n$ of the *n* examples, $X_1, X_2, \ldots X_n$ such that for all i, D_i describes X_i. The general CAL segment, *C** can then be applied to the example descriptions, $D_1, D_2, \ldots D_n$ to produce the desired exercises $E_1, E_2, \ldots E_n$. The general CAL segment, C* is an abstract treatment of the lesson material, as it does not refer to specifics until applied to the concrete examples specified by the example descriptions. Hence the name *'courseware abstraction'*. The one general lesson is abstracted from specifics and then made concrete through application to a set of specific example descriptions.

It is important to note that the descriptions $D_1, D_2, \ldots D_n$ may either be explicitly created by the lesson author in the form of a database which defines each of the relevant attributes of each example, or may be implicitly defined in the form of a sub-routine which generates examples and descriptions of their relevant attributes as required.

The most immediately apparent advantage of courseware abstraction is the potential for reducing authoring time. Given a cost of *x* for creating each CAL segment under the direct approach, the overall development cost of a lesson will be *nx* (where *n* is the number of examples provided). Thus the cost per exercise will remain constant at *x*, no matter how many exercises are created. (Certainly there will be some minor time saving through the ability to copy lesson sub-segments from one similar exercise to the next, but these will be largely offset by increased problems with ensuring that all necessary alterations occur to adjust the CAL segment to the new example.) In contrast to the direct approach, given a cost of *y* to create the single lesson segment *C** and *z* to specify each example description, the total lesson cost will be $y + nz$. From this it follows that the cost per exercise will be $y/n + z$. Thus, the cost per exercise will decrease as the number of exercises produced increases.

It is argued below that the costs of producing an example description can be negligible, given an appropriately designed lesson. Given this, the cost of an abstracted lesson will be virtually constant, irrespective of the number of exercises produced. It will almost always be the case that the time taken to produce a CAL segment directly examining an example, *X*, will be substantially greater than it takes to produce a description of *X*. A description will only require a list of the relevant attributes of *X*. A CAL segment will have to manage whatever interactions with the student are necessary for the exercise examining *X*.

It follows that even if the one-off cost of producing the abstracted lesson is substantially higher than that of producing each direct CAL segment tailored to a specific example, so long as sufficient examples are to be examined, courseware abstraction will be markedly cheaper. For example, to consider only lesson authoring time, supposing that each direct CAL segment takes 1 hr to produce, that the abstracted lesson takes 50 hrs to produce and that each example description takes 1 min to produce. (These figures will obviously vary greatly from lesson to lesson.) Using these example figures, a lesson containing 100 exercises will take 100 hrs to produce under the direct approach and just 51 hrs 40 mins using courseware abstraction. Clearly, the number of exercises needed before courseware abstraction requires less development time than the alternative will vary substantially from lesson to lesson. However, for any lesson, if sufficient exercises are provided, then eventually the balance will tip in favour of courseware abstraction.

Furthermore, given the lower cost of producing a new exercise once a lesson has been created using courseware abstraction, the lesson author is likely to be far more inclined to produce additional exercises than if they were using the direct approach. Using the direct approach there will be a strong motivation to produce the minimum possible number of exercises. Using courseware abstraction there will be every incentive to produce as many exercises as may possibly be beneficial to the student.

However, reduced authoring time and facilitating the provision of as many examples as may be desirable are not the only advantages of courseware abstraction. It also aids lesson verification. Inevitably, when a lesson is created, various errors will be incorporated, ranging from typographical errors to factual errors. It is very important to thoroughly test a lesson before it is presented to students, so as to uncover and remove any such errors. It is highly undesirable to teach the student erroneous material, even if it does arise from a simple arithmetic error on the part of the author!

Under the direct approach, lesson verification inevitably required the examination of every aspect of every exercise. For example, a spelling error could occur in CAL segment associated with any aspect of any exercise. In contrast, to verify an abstracted lesson it is only necessary to verify the one general lesson and a body of example descriptions. So long as the example descriptions are recorded in a reasonable format, their verification will be easy to achieve. Figure 1 shows the form that such an example description takes in the *DABIS* system, a CAL system that supports courseware abstraction (Webb, 1986b, 1988). This is the only example-specific information that is required by the system to provide comprehensive lessons on each example. Specific treatments are then created by app1ying the abstracted lesson to these specific descriptions. It should be apparent that such example descriptions can be readily created and verified at little cost in authoring time.

2.1 Determining the relevant issues for an example

Although the general mechanisms for its implementation are straightforward, there are a number of difficulties that must be overcome by the lesson author using courseware abstraction. One such problem is the identification and correct treatment of the different issues that relate to different examples from one domain. For example, when examining simple arithmetic problems it will be necessary to determine, among other things:

— the operator to be applied;
— the result of single digit addition on each column;
— the amount to be carried to each column, and/or
— the result of single digit multiplication of each column.

However, not all of these issues will necessarily be relevant to each example For instance, when examining a problem using the *multip1ication* operator it will not be necessary to determine the result of the single digit *addition* of each column, and vice versa.

In general, when examining a set of examples, there will be a set of issues, $\{S_1, S_2, \ldots S_m\}$ which will need to be examined. During any exercise E_i examining an example, X_i there will be a set of relevant issues, R_i, such that R_i , $\{S_1, S_2, \ldots S_m\}$. A difficulty faced by lessons using courseware abstraction is how to determine which issues relate to a given example, and thus, which portions of a lesson should be applied to it. To this end it is necessary for lesson descriptions to contain not only a general description of the example, but also information that enables the issues that apply to that example to be identified. As can be seen from Fig. 1., the DABIS system achieves this by associating a set of attributes with each example. This enables the system to identify not only the relevant is but also the correct treatments of those issues for each example.

DABIS demonstrates the ease with which it is possible to determine both the issues that are relevant to a particular example and the correct treatments of those issues when example descriptions are created explicitly by the lesson author. In contrast, these problems are major obstacles to the use of computer-generated examples in courseware abstraction. In standard CAL environments the issues that pertain to each example can be readily specified by a simple

mechanism, such as the use for each issue of a simple Boolean variable that flags whether that issues applies to the current example. This simple device enables the relevant aspects of a lesson to be immediately available to that lesson.

Instance:	is
Attributes	{copula, verb}
sentence	She is an old woman.
word class	verb
verb	is
noun	woman
pronoun	she
preposition	@@@@@@@@
determiner	an
adjective	old
adverb	@@@@@@@@
Instance:	that
Features	{demonstrative, determiner}
sentence	Give that money to me immediately
word class	determiner
verb	give
noun	money
pronoun	me
preposition	to
determiner	that
adjective	@@@@@@@@
adverb	immediately
Instance:	fell
Features	{intransitive, verb}
sentence	He fell under a speeding truck
word class	verb
verb	fell
noun	truck
pronoun	he
preposition	under
determiner	a
adjective	speeding
adverb	@@@@@@@@

Figure 1. Some sample example descriptions.

This listing was created by the *DABIS* lesson description program (Webb, 1986a). The first entry for each example is a short description (in this case a word.) The next entry is a list of relevant attributes that the word exhibits. Subsequent entries provide short pieces of text called text macros. The text macros in this lesson describe a sentence in which the word appears, and words of different word classes that appear in that sentence. @@@@@@@@ indicates that no word with the relevant word class appears in the sentence.

3 Courseware Abstraction and Student Evaluation

One important aspect of student evaluation is determining which issues the student has mastered and which are causing difficulty. This can be very difficult to achieve using the direct approach to CAL. A mechanism must be created that enables treatments of one issue to be identified from one exercise to the next. Establishing the identity of treatments of issues across CAL segments is far from trivial. For example, if a student has difficulty in arithmetic with carrying digits from one column to the next, then all of the student's failures at the particular points that happen to deal with carrying digits in each exercise must all be related to one another for proper identification of the error to occur.

In contrast, courseware abstraction is not faced with this problem. Because only one CAL segment is applied to all examples, it is trivial to keep track of the identity of each treatment of an issue from exercise to exercise. Quite simply, the treatment of an issue is always the same portion of the same CAL segment, irrespective of which exercise is considered. This greatly facilitates student evaluation. Very useful information about student understanding can be provided simply by collection statistics on student performance in each portion of a lesson across the entire set of examples that a student examines. The ECCLES system (Richards & Webb, 1985) uses exactly this strategy to produce very detailed student assessment, without any need for prior allowance of any form by the lesson author. For example, if a student invariably makes errors in the portion of a lesson that pertains to multiplication of single digit numbers then it is likely that they require particular attention in the area of single digit multiplication.

More sophisticated student analysis can be produced by the use of more sophisticated analysis of the interactions between different attributes of the examples that a student examines and the errors made (Webb, in preparation).

3.1 The facilitation of factually complete lessons

Following the direct approach to lesson authoring, every issue that relates to each example will require separate explicit treatment by the author, for each example to be examined. This can lead to an extremely heavy authoring load. The author's response to this situation is likely to be threefold:

1. To avoid examining more than the very minimum number of examples;
2. To avoid examining more than the bare minimum number of issues for each example, and
3. To abbreviate each treatment of each issue to the greatest possible extent.

In contrast, no such pressures bear upon the author of an abstracted lesson. As already discussed, the psychological inclination is likely to be, if anything, toward producing a large number of examples.

Furthermore, two factors are likely to encourage the development of factually exhaustive lessons - lessons that examine all issues in detail. The direct approach to lesson authoring requires the author to create a separate treatment of each issue for each example. As a result there is a large pressure on the lesson author to minimize the extent of each treatment of each issue, as the effort devoted to it will be multiplied by the number of examples examined. By comparison, using courseware abstraction, the lesson author knows that a treatment of an issue will only have to be created once and thus is under less pressure to limit the effort put into its construction. The second factor encouraging the development of factually complete lessons arises from creating lessons in the abstract, that is, outside of the context of a specific example. As a result, the issues to be examined, their interrelations, and the interrelations of the CAL segments dealing with them, should be very clear, as they will not be obscured by the presence of specific details relating to individual examples. This should lead to the development of more rigorous and theoretically sound lessons.

3.2 Example selection

When presenting a large number of examples to a student, it is desirable to select examples that are most relevant to a student's current understanding of the domain. By explicitly separating the

examples to be examined from the CAL treatments through which they are examined, courseware abstraction greatly facilitates the individualized selection of appropriate examples for each student.

A particularly powerful method for managing this process is enabled by the explicit identification of the issues with which a student experiences difficulty, and the issues which relate to each example that is available to the system. Quite simply, when it comes to selecting a new example, one of the factors that a system can accommodate is the number of issues with which the student is experiencing difficulty that relate to each example. Examples that only involve issues which the student has fully mastered are unlikely to be beneficial, as are examples which involve large numbers of issues with which the student is experiencing difficulty. Allowance for this factor can greatly aid the provision of appropriately adaptive lessons. In a system that stores a library of available examples in an external file, example selection can involve searching that file for an example to which an appropriate combination of issues relates. Examples for which the student has mastered all issues can be rejected, since the student's knowledge will not be extended by their consideration. The ideal example to select is the one with the largest numbers of issues mastered but for which at least one issue is not mastered. Examination of such an example will provide the student with the maximum opportunity to improve her/his understanding of a poorly understood issue.

This very simple technique can readily adapt instruction, so that i is closely aligned to each student's aptitude and understanding in a domain. For the interested readers, a more sophisticated approach to example selection in courseware abstraction is provided by the DABIS system (Webb, 1986b).

It is important to note that the identification of relevant issues and; the means to evaluate the student's understanding of each is necessarily a problem that must be confronted by the lesson author on a lesson by lesson basis. General automatic techniques are not yet available.

4 Generative CAL and Courseware Abstraction

It has already been mentioned above that many CAL lessons have previously used courseware abstraction. It also appears that many systems have relied heavily on the advantages of courseware abstraction for much of their power. It is perhaps surprising in this light that courseware abstraction has not previously been identified as an important CAL methodology.

Generative CAL is a prime example of an approach to CAL that has exploited the power of courseware abstraction without recognizing the of courseware abstraction as the true source of that power. Generative CAL is the provision of a CAL lesson that generates examples that are then examined by a general lesson (Uttal *et al.*, 1969; Uhr, 1969). In other words, it is the use of courseware abstraction with computer generation of example descriptions. The literature has focused upon the generation of examples as the important feature of generative CAL. However, this is not the aspect of generative CAL which provides its power. After all, the outcome of generating an example is identical to the outcome of selecting that example from an explicitly defined set of examples. Indeed, the generation subroutine in a generative lesson can be viewed as an implicit description of a set of examples - the set that the function is able to generate.

The only possible advantages of generating the examples rather than selecting them from a body of explicit descriptions are that:

— it may not be feasible to explicitly specify the full set of examples that a generative system has implicitly defined, and
— for practical reasons, such as computational or authoring convenience it may be more efficient to generate rather than to explicitly define examples.

However, in most cases this latter benefit is likely to be more than offset by the difficulty of creating a subroutine that is capable of generating the desired examples and the necessary information about the their attributes. In practice this means that the computer must be able to solve the problem that it generates. It is no coincidence that the majority of generative CAL systems have been in the domain of mathematics. This is the one area in which it is relatively simple for the computer to generate examples and determine their relevant attributes.

Despite the importance of courseware abstraction for generative CAL, no major previous treatment of generative CAL (or indeed on any other aspect of CAL) has identified the separation of general courseware from explicit examples as an important technique. The emphasis in the literature on the generation of examples, rather than the selection of appropriate examples, as the primary feature of generative CAL, has possibly done much to limit the development of CAL during the seventies and eighties. Generating the detailed information required for examples in non-trivial domains is, in most cases, extremely difficult. In contrast, it is simple to create explicit example descriptions to which abstracted lessons can be applied. If courseware abstraction, rather than generative CAL, had been emphasized in the late sixties, then CAL research and development may have taken quite a different course during subsequent years.

Generative CAL is not the only CAL paradigm to have exploited courseware abstraction. Most intelligent tutoring systems also utilize it. Prominent examples include *SOPHIE* (Brown, Burton & deKleer, 1982), which generates examples of electronic circuit faults; *GUIDON* (Clancey, 1987), which selects examples from a set of medical case histories and the *LISP* tutor (Anderson & Skwarecki, 1986), which selects examples from a set of pre-specified *LISP* programming problems. It is interesting when considering these and other similar systems to analyse exactly how much of their power arises from their use of courseware abstraction.

5 Implementing Courseware Abstraction

Courseware abstraction can be used to create lessons in most existing CAL environments. The minimum that is required for application of the technique is:

— the ability to use variables that may assume different values in different contexts;
— the ability to assign values to those variables depending upon the example to be examined, and
— the ability to alter the interactions with the student depending upon the values of those variables, for example, through conditional branching.

The example descriptions in Fig. 1 primarily utilize simple text values for a set of simple parameters. The exception is the *attributes* parameter which effectively lists the issues that relate to the example. There is no reason why a particular application of courseware abstraction should limit itself to the use of text parameters, however. Different subject areas are likely to suggest different forms of parameters. For example, lessons in arithmetic are likely to use numeric parameters, while lessons in phonetics are likely to require representations of different sounds.

One approach that could be used to implement the required mechanisms would be to define internally to the lesson a set of variables of appropriate types for each of the lesson parameters. For each exercise it would be possible to read in a value from an external file for each parameter. The lesson author could then create the set of available examples by inserting relevant details for each example in that file. Each time that the lesson required another example, it could select one from those provided by the lesson author and read the relevant details into the internal variables from the file.

This should not be taken to be a suggestion that this is the best possible mechanism for implementing courseware abstraction. It should be clear that many different mechanisms could equally well be applied. As discussed above, the generation of examples with the desired attributes is another technique that has already been used with a great deal of success. Another possibility is the use of explicitly defined examples for which the appropriate details are calculated by the CAL system, as needed.

6 Conclusion

Courseware abstraction is a CAL technique that, although widely used, has not previously received explicit recognition. The technique has numerous advantages over alternative approaches to lesson creation in situations where it is desirable to examine a large number of generically related examples:

— it is cost efficient;
— it facilitates lesson verification;
— it encourages the provision of large numbers of examples where these are desirable;
— it simplifies the selection of appropriate examples for presentation to each student;
— it provides a convenient framework for student evaluation, and
— it supports the development of factually exhaustive lessons.

To further summarize these features, courseware abstraction offers the unusual partnership of reduced overheads coupled with qualitative improvements!

Both generative CAL and many intelligent tutoring systems make use of courseware abstraction. However, recognition of its contribution has not been forthcoming from within either of these paradigms. Indeed, the focus on the generation of examples in generative CAL has been misguided and has directed attention away from the real source of its power - its use of courseware abstraction.

Acknowledgements

The research presented in this paper grew out of the *ECCLES* project which was established and led by Tom Richards. Thus, the ideas presented here have directly evolved from Tom's original inspiration for *ECCLES* .The ideas presented herein were developed while the author was at the Department of Computer Science, La Trobe University and at the School of Computing and Information Studies, Griffith University.

References

[1] Anderson, J.R. & Skwarecki, E. (1986) **The automated tutoring of introductory computer programming**. *Communications of the ACM*, 29, 842.

[2] Brown, J.S., Burton R.R. & deKleer, J. (1982). **Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III.** In *Intelligent Tutoring Systems* (eds. D. Sleeman & J.S. Brown) pp. 227, Academic Press, London.

[3] Clancey, W.J. (1987) *Knowledge-Based Tutoring: The GUIDON Program*. MIT Press, Cambridge, MA.

[4] Richards, T.J. & Webb, G.I. (1985) **ECCLES: An 'expert system' for CAL**. In *Proceedings of the 1985 Western Educational Computing Conference*, pp. 151. Oakland, CA.

[5] Uhr, L. (1969) **Teaching machine programs that generate problems as a function of interaction with students.** In *Proceedings of the 24th National ACM Conference*, pp. 125.

[6] Uttal, W.R., Pasich, T., Rogers, M. & Hieronymus, R. (19691. **Generative Computer Instruction**. *Communication* 243, Mental Health Research Institute, University of Michigan.

[7] Webb, G.I. (1986a) **The Domain-Analysis Based Instruction System Reference Manual**. *Technical Note* 3/86, Department of Computer Science, La Trobe University, Melbourne.

[8] Webb, G.I. (1986b). **Knowledge representation in computer-aided learning: the theory and practice of knowledge-based student evaluation and flow of control**, *PhD thesis*, School of Mathematical and Information Sciences, La Trobe University, Melbourne.

[9] Webb, G.I. (1989) **A knowledge-based approach to computer-aided learning**. *International Journal of Man-Machine Studies*, in press.