

THREE PROGRAMMING BEANS

Gustavo Moratorio
London South Bank University
London SE1 0AA
moratog@lsbu.ac.uk
<http://cise.lsbu.ac.uk/pooples>

Fintan Culwin
CISE
London South Bank University
London SE1 0AA
fintan@lsbu.ac.uk
<http://cise.lsbu.ac.uk/pooples>

ABSTRACT

E-learning has so far failed to deliver what it has promised in terms of improvement to the student experience. The focus in recent years seems to have been placed mostly towards electronic content management and delivery, and less towards students' activities. Concepts from Object Oriented (OO) programming have been borrowed to create Learning Objects. These objects lack instructional and interactional flexibility and tend to possess content dependencies which compromise their reusability. Software beans are examples of OO frameworks which separate activity management from content management. When this is incorporated at the right level of granularity, these Learning Objects gain these qualities and are better described as Learning Beans (LBs).

Keywords

Learning Objects, Learning Beans, Learning Frameworks.

1. INTRODUCTION

E-learning from an organisational perspective, has the dual purpose of reducing the costs of instructional material delivery whilst improving the quality of the learner's experience [1]. However little of these dual benefits have been delivered.

From a financial point of view and for many institutions, the failure to reduce costs largely results from the use of such materials as an extension to existing systems rather than their replacement. From an instructional point of view, Managed Learning Environments (MLEs) such as Blackboard, WebCT or MOODLE, provide excellent

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

© 2006 Higher Education Academy
Subject Centre for Information and Computer Sciences

tools for both the management of such content and its delivery from tutors to students, but these often fail to enrich student learning activities. These systems are also characterized by their high levels of granularity.

Recent years have seen the emergence of Learning Objects (LOs). Borrowing concepts from Object Oriented Programming (OOP), learning objects are characterised by their low levels of granularity. However, contrary to the principles of OOP, learning objects are considered rigid both in their instructional and interactional design, and in the content they deliver. This lack of flexibility is often due to instructional designers embedding aspects of their own subjective learning theory and other pedagogic baggage.

Aside from LOs low granularity, there seems to be a large amount of confusion about their defining characteristics. The IEEE [2] defines LOs as '...any entity digital or non-digital which can assist in learning or teaching...'. This exemplifies the broadness of many definitions and the lack of agreed specific defining characteristics.

Ideally, LOs should be philosophically and pedagogically neutral. However this is recognized as not being an entirely realistic goal. The design of any artefact reflects the culture and politics of the environment in which it is produced [3].

Learning can be described as a gradual activity which consists of a cyclical progression between skill levels. In the early stages of learning to program, several specific low level cognitive skills need to be developed. Although the order in which skills relating to sequence, selection and iteration are learnt varies depending upon the tutor's philosophical and pedagogic position; their importance is undisputed. However, it has been recognized by many studies [4] that students in mass fail to develop these skills.

Constructivist theories state that learners develop their own understanding by continuous practice and engagement with the material. Purnell [5] divides learning in four levels of competence. In their initial stage, learners are unaware of both their capability

and the subject's demands. This is described as unconscious incompetence. The learner quickly becomes aware of his/her level of incompetence and enters the next level, described as conscious incompetence. Further engagement with the subject area allows learners to carry out various tasks at an acceptable level of competence. However, such performance requires the learner's full attention. This level is known as conscious competence. The last stage is unconscious competence. At this stage the learner can perform tasks at an acceptable level of competence with minimal concentration. The gradual nature of learning is also recognized by Bruner [6] who proposed learning as a spiral process in which deeper understanding is gained with each interaction with the subject area.

This paper will describe three proof of concept programming beans (imperative, selection and loop). The beans framework emphasises LOs as being context unaware, of low granularity and philosophically and pedagogically neutral. The framework has also been designed to recognize that learners will need repeated interaction with the bean, each time needing a more complex challenge. The design also recognizes the need for instructional designers to manage the learners' challenge level and provides a content independent configuration tool to facilitate a bean's progressive disclosure.

2. THE LEARNING BEAN FRAMEWORK

Borrowing from the principles of Java Beans, Learning Beans (LBs) can be described as a framework into which instructional beans can be plugged. LBs possess the flexibility and instructional content independence not found in LOs. The intention was to abstract the content free behaviour and place it into a framework where content specific components can be embedded and managed.

The LB context recognizes three roles. Learners interact with bean instances in order to attain and demonstrate skills. Instructional object designers design and build a learning object bean that can be plugged into the framework. Instructional designers create and configure instances of the bean and place them within an instructional context suited to the learner.

The learning beans framework allows instructional designers to select a particular behaviour from a set of possible options to suit different skills levels, so allowing progressive disclosure of the material. The behaviour selection is carried out by the instructional designer via the configuration utility, during the creation of a new learning bean instance. The content free configuration utility is shown in Figure 1.

The configuration utility is divided in two well defined parts. The first being the content free

configuration, which specifies the behaviour for any bean that is to be plugged into the framework.

Bean configuration	
Challenge	true
No. of tries	3
Reveal	true
Reveal limit	4
TimeOut	true
Time Allowed (mins : secs)	0 : 3 0
Summative	true
Marking	Variable
Fixed Points	3
Variable Points	10 5 3
<input type="button" value="Configure"/> <input type="button" value="Reset"/>	

Figure 1 LB's Content Free Configuration Utility.

Describing the controls from the top to the bottom: The *Challenge* control determines if the user is challenged to perform a particular task or not. If challenge mode is not chosen then all remaining options are disabled and the user will be allowed to explore the bean in an unstructured manner.

The *No. of tries* control determines the number of attempts that the user is allowed to solve the challenge. Setting the value to zero allows an unconstrained number of tries. The *Reveal* control determines if the user is allowed access to a solution to the challenge, or not. This control can be used in conjunction with the *Reveal limit* control which determines how many attempts the learner has to make before a solution is made available.

The *Time Out* control determines if a time limit should be applied, or not. If a limit is requested then the *Time Allowed* control can be used to indicate how long the user should be allowed. The *Summative* control determines if the bean is to be presented in summative or the default formative mode. If the bean is in summative mode the remaining controls determine the allocation of marks.

The *Marking* control determines if a fixed number of points are available or if the number of points varies according to the number of attempts. The *Fixed Points* and *Variable Points* controls specify the number of points accordingly, with an arbitrary limit of five different values for variable marking. The initial version of this control allowed an indefinite number of values, at a significant cost to the clarity of the user interface. After reviewing this facility, it was decided that although the intention was to provide support for any pedagogy, this was not to be at the cost of pedantry and five values were sufficient.

These additional controls might allow a configuration where a learner using the bean in a

summative environment is allowed three attempts within 2 minutes and awarded 10, 5 or 3 marks respectively should they solve the challenge on the corresponding attempt. A large variety of other possible configurations are also available.

The bean configuration utility is presented to the user with a sensible set of defaults and dynamic behaviour that guides the instructional designer. For example the *Time Allowed* control is disabled if the *Time Out* control is set false. Other logical impossibilities are also detected, for example specifying five variable points whilst only allowing three tries, and result in a message dialog to the user when the *Configure* button is pressed.

When configuration is complete and acceptable, pressing the *Configure* button will result in an instance of the bean being launched in a separate window. The instructional designer can use this instance to verify that its behaviour is as they intended. Also available in the window is the HTML code that can be copied and pasted into a Web page to make the bean deployable in its intended environment.

3. PROOF OF CONCEPT BEANS

Three proof of concept programming beans have been plugged into the above framework. For reasons of space, this paper will describe only the imperative bean. However all beans can be seen in Figure 2.

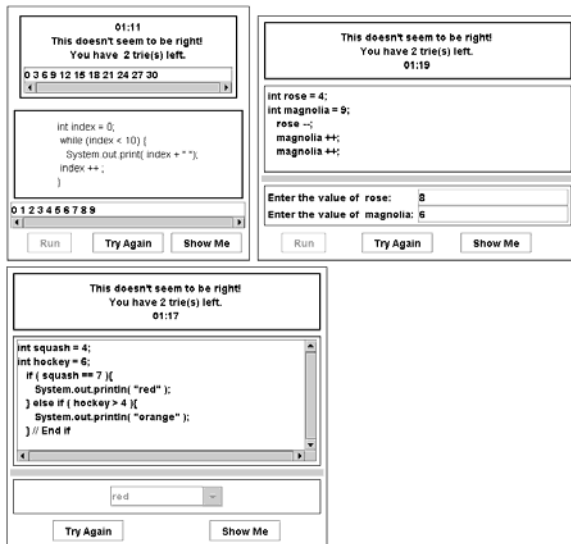


Figure 2 Loop bean (top left), imperative bean (top right) and selection bean (bottom)

The imperative bean aims to aid students on the first level of software development. The skills required at this level are found very difficult by students, and yet regarded as essential. In the example show above, two variables (*rose* and *magnolia*) were declared and the values 4 and 9 respectively assigned to them. The student is asked

to predict the resulting values of each variable after the execution of the three lines shown. In this example the first line increments *rose* by 1 (*rose++*;) The second and third both increment the value of *magnolia* by 1 (*magnolia++*). The student is asked to predict the resulting values (5 and 11 respectively).

The bean content specific behaviour can be configured to other more complex settings where a wider range of features become available illustrating the beans' progressive disclosure capabilities and adaptability to students at different skill levels. This is done via the subject specific section of the configuration utility as shown in Figure 3.

Imperative Bean configuration	
Negative Values	false
Number of Variables	2
Number of Lines	3
Maximum Declaration Value	10
Maximum Mult/Div Value	5
Maximum Modulo Value	11
Capability	POST ONLY
Bean configuration	POST ONLY POST-PRE ADD-SUB ADD-SUB-SELF ALL-BUT ALL-MODULO DIV-MOD ALL-BUT-SELF ALL-MOD-SELF EVERYTHING
Challenge	
No. of tries	
Reveal	
Reveal limit	

Figure 3 Imperative bean content specific configuration utility

The first control *Negative Values* determines if any of the initial, intermediate or final values of the variables are ever allowed to be negative. Having to deal with negative values adds to the cognitive complexity of the challenge. The *Number of Variables* control determines the number of variables declared and subsequently manipulated. The range is from 1 to 3 and having to deal with more variables adds to the cognitive complexity of the challenge. The *Number of Lines* control, range 1 to 5, determines the number of lines of code that follows the declarations; with more lines adding to the cognitive complexity.

The *Maximum Declaration Value* control determines the maximum initial value of the variables. The choices are 10, 25, 50, 75 and 100; with larger values having greater cognitive complexity. The next two controls *Maximum Mult/Div Value* and *Maximum Modulo Value* are only available when certain *Capability* choices are made; as described below.

The *Capability* control is the major determiner of cognitive complexity as it specifies which operations are allowed upon the variables.

The simplest capability is the post increment operation only, as described above. The next possibility defines post and pre increment and decrement possibilities (++rose; rose++; --rose; rose--). ADD-SUB introduces the concept of assignment and the addition and subtraction operations. For example the line: *tom = dick + harry*; has the meaning take the value of the variable *harry*, add it to the value of the variable *dick* and place the resulting value into *tom*.

The self-referential variants of the addition and subtraction operations have the form: *tom += 5*; which has the meaning add 5 to the current value in *tom*. The ALL-BUT-MOD capability allows all four simple binary operations addition (+), subtraction (-), multiplication (*), integer division (/). The next possibility, ALL_MODULO, adds integer modulo division (%). The difference between simple and modulo division being that the simple integer division of 13 by 5 gives 2; as 5 goes into 13 2 times. Modulo division gives the remainder after division, 3; as 5 goes into 13 2 times remainder 3. It is at this point in the table that the controls *Maximum Mult/Div Value* and *Maximum Modulo Value* controls become available. The remaining two options in the list should be comprehensible from the descriptions already given.

4. EVALUATION

Two small evaluation studies have been conducted. The first involved a group of eight second year computing students starting a core Java based unit. The second study involved nine final year business information technology students taking an elective C++ programming unit.

Both groups had the same pattern of experiences starting with an unseen pre-test. This was followed by use of the programming beans, a distraction activity and a post test which was comparable with the pre-test.

The tests consisted of two tasks each in sequence, selection and iteration; being paper versions of the challenges that the beans would deliver. The first of the two tasks was set at a minimal level of difficulty, with the second set at a moderate level of difficulty.

Each pair of tasks was followed by a five point Likert scale measuring the confidence that the students had in their answers. The second group had an additional scale for the pre-test only asking them to indicate if they thought that they should be able to solve problems like these. This change to the protocol came about because of a comment by a student on the first investigation who said "Of course we should be able to solve problems like this"

	task 1	task 2	average	conf
pre	0.65	0.35	0.50	2.76

post	0.97	0.94	0.96	4.35
Delta	0.32	0.59	0.46	1.59

Table 1 Sequence Results

Table 1 shows the outcome for the sequence exercises. The pre and post task scores are out of 2.0 and show, for example, that the average pre task score on the simpler first task was 0.65, the average post task score on the more complex second task was 0.94 and the average post task score on both tasks was 0.96. The last column gives the confidence score out of 5, with larger numbers indicating greater confidence. The delta row is the difference between the pre and post measurements.

	task 1	task 2	average	conf
pre	0.53	0.29	0.41	3.12
post	0.94	0.65	0.79	4.12
Delta	0.41	0.66	0.38	1.00

Table 2 Selection Results

Tables 2 and 3 present the corresponding data from the selection and iteration tasks respectively. In all three cases the data shows a distinct improvement in performance between the pre and post tasks.

	task 1	task 2	average	conf
pre	0.24	0.18	0.21	2.00
post	0.71	0.53	0.62	3.82
Delta	0.53	0.35	0.41	1.82

Table 3 Iteration Results

More significantly, there is a distinct jump in the students' self reported confidence in being able to do these tasks. Self confidence is very significant as it is a valuable tool to counter the learned helplessness which is often the factor which blocks progression from conscious incompetence towards conscious competence.

	pre test		post test		de lta	
	Abil	conf	abil	conf	abil	conf
seq C	0.3	2.5	1.0	4.4	0.7	1.9
seq J	0.6	3.0	0.9	4.3	0.3	1.3
sel C	0.4	2.9	0.8	4.4	0.4	1.5
sel J	0.4	3.3	0.8	3.9	0.4	0.6
loop C	0.1	1.6	0.7	4.0	0.6	2.4
Loop J	0.3	2.3	0.6	3.7	0.3	1.4

Table 4 Inter group comparison

Table 4 presents the essential data from the less technical C++ group, identified as C, and the more

technical Java group, identified as J. In all cases there is a noticeably larger increase in confidence for the less technical group. This is also the case for ability, apart from selection where the increases are equal.

The C group were also asked to indicate if they thought that they should be able to perform the tasks set. The results for this question were: 3.88 for sequence, 3.12 for selection and 2.00 for iteration. This seems to indicate that pre task self confidence for iteration was noticeably lower which makes the post test confidence measure for this task particularly significant.

However; this was a small scale, limited duration study conducted in the institution and by the team who developed the beans. There is every possibility of a Hawthorne effect and the longer term benefit of having access to these facilities is unknown. Observationally one student from the J group who was noticeably weak was offered further access to the resources to address the weakness. The student did not take up this offer explaining that he did not see those skills as being essential to passing the unit.

5. CONCLUSIONS

The production of low granularity objects that are flexible in their pedagogic content and instructional capabilities, whilst inexpensive to produce can be considered advantageous when compared with most other LOs. This advantage is mainly obtained by the separation of the content specific material from the activity management aspects.

The framework is firmly grounded in specific, named learning theories. The design provides an interactive approach in which the learner must actively engage with the application. The provision of dynamically changing challenges results in learners acquiring a higher level of understanding. Learners can not merely achieve the correct answer by shallowly remembering its result.

The design caters for progressive disclosure recognising that learners pass cyclically through different stages. In this way a student can gradually gain a deeper level of understanding by using the same bean with a different content specific configuration upon each revisit. Instructional designers are able to select the level of difficulty required to suit a particular learning path.

One feature that has been suggested as missing from the bean framework is an 'explain' capability. This feature would show the learner what a solution

to a challenge might be and explain why the solution is a correct answer. However, this feature is believed to make the bean context dependant. By

passing the responsibility of explanation design to the instructional object designer, the bean will no longer have context flexibility, as the explanation embodies a conceptual position which is embedded within the bean.

6. ACKNOWLEDGMENTS

The initial development of the beans was supported by a learning & teaching fellowship grant from the faculty of Business, Computing and Information Systems (BCIM) at London South Bank University (LSBU). Later development was supported by the Higher Education Academy subject Centre for Information & Computer Science (HEA/ICS) and LSBU. The beans are freely available at the URL given at the start of this paper and at the HEA/ICS website and have also been archived in JORUM [7].

7. REFERENCES

- [1] Dearing R. (1997), Higher education in the learning society, London: The Stationary Office ISBN 0104023988
- [2] IEEE (2202), IEEE Standard for Learning Object Metadata, available at <http://ltsc.ieee.org/wg12/part1484-12-1.html>, Document dated Nov 2002 (Accessed 10 April 2006)
- [3] Jürgen L. (2001) Issues in agent-oriented software engineering, First interational workshop, AOSE 2000 on Agent-oriented software engineering, Limerick Ireland
- [4] McCracken M. et. Al, (2001) A multi-national, multi-institutional study of assessment of programming skills of first year CS students, ACM SIGCSE Bulletin Volume 33 Issue 4 2001
- [5] Purnell, L. D. (2002) The Purnell's model for cultural competence. Journal of Transcultural Nursing
- [6] Bruner J. S. (1974), Toward a Theory of Instruction, Harvard University Press ISBN 0674897013
- [7] JORUM The JISC Online Repository for Learning and Teaching Materials, www.jorum.ac.uk/ (accessed 10 April 2005).