Learning Beans: Design, Implementation & Evaluation

Fintan Culwin

CISE

London South Bank University

London SE1 0AA, UK

+44 (0)20 7815 7434

fintan@lsbu.ac.uk

ABSTRACT

This paper describes the use of statechart notation to design the pattern of behavior that a user will have when interacting with a learning bean. The larger context of a learning bean and the stakeholders that are involved in its deployment and use are also introduced, two beans are briefly described and the results of a user trial presented.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques *modules and interfaces, object-oriented design methods, State-diagrams, user interface, human factors.*

General Terms

Algorithms, Design, Human Factors, Standardisation.

Keywords

learning objects, learning beans, learning activity management systems (lams), learning frameworks, usability.

1. INTRODUCTION

Learning objects have many conflicting definitions [1]. One consistent element of the definitions is that they have low granularity and so are focused upon one simple learning outcome. This is in accord with software objects whose intention was that object re-use would allow applications to be assembled like Lego bricks. This turned out not to be the case for the first generation of software objects [2] and has also not been the case for learning objects. Reuse is particularly important for educational resources as they can be very expensive to produce and tend not to be used outside the immediate environment they were developed in.

Second generation software objects, such as Java beans [3] promised greater potential reusability by defining a generic behavioral framework into which application specific objects could be placed. Learning beans [4] are an attempt to transfer this concept to the domain of learning objects. The learning bean behavioral framework provides a large range of patterns of possible learner interaction into which content specific learning objects can be plugged and specific patterns of interaction defined. This is comparable to Learning Activity

Management Systems (LAMS) [5] which, at a much higher level of granularity, separate the activities involved in learning from the content of what is learned.

The behavioral framework implemented in a learning bean is pedagogically promiscuous. That is; they are not wedded to any particular theory of how people learn, rather they are capable of being configured to implement a variety of particular behaviors within the framework.

The usability of the beans is central to their successful deployment and with such a low granularity they would have to have near walk-up capability. That is a learner should be able to interact with the bean efficiently and effectively with little or no instruction. In order to assure this requirement at the design stage the behavioral framework was designed using statechart notation [6]. Once this design had been completed and verified, the software interface between the framework and the content objects could be finalised.

The wider context of the bean's usability was also considered and three stakeholders were identified. The learner would need walk-up usability with a deployed bean. The learning interaction designer would be responsible for configuring and deploying the bean within a learning context. The learning object developer would need to design a separate, highly interactive, content bean which, by means of the software interface, could be plugged into the framework to produce a completed learning bean.

2. THE FRAMEWORK DESIGN

A slightly simplified design of the learning bean behavioral framework is presented using state transition diagram notation in Figure 1. A state transition diagram consists of states, shown in round edged boxes, and transitions, shown as labeled arrows connecting two states. The label is in two parts shown above and below a line, either of which may be omitted. The label above the line is an event which must occur for the transition to be considered. An empty event label means that the transition will be continually considered. The label below the line is the pre-condition which must be true for the transition to be followed. An empty pre-condition label is always true.

The initial state is shown as a solid circle, from which in this design there are two possibilities. If the bean is in *timed* mode it enters the *Timed Start* mode where the learner can use a *start* control to enter the *Content Bean* state. This is also the initial state when the bean is in the simpler *challenge* mode.

In the *Content Bean* state the learner will interact with the embedded interface until they are ready to *try* their solution to the challenge. When the user *tries* the outcome can be correct or incorrect, leading to either the *Solution Incorrect* or *Solution Correct* state respectively. From the *Solution Incorrect* state there are four possible transitions, the user can *try again*,

Figure 1 Behavioural Framework STD

if allowed, re-entering the *Content Bean* state. Alternatively the user can ask to *show* the solution, if allowed, and enter the *Solution Shown* state. If the learner has run out of tries and is not allowed to see a solution then they will be informed of this, and they can transit to the *Out of Tries* state by acknowledging the message with the *ok* control. The fourth transition can also be followed from the *Content Bean* state, as shown by the dashed superstate. This is the *time out* transition which can only occur if the interface is in timed mode and which leads to the *Timed Out* state. From the four possible states that lead from this superstate, the learner can request *another go* at a different challenge, if allowed, entering the *Content Bean* state again.

The textual description of the design given above is implicit in the notation. A design commentary would clarify the possible paths that could be followed through this behavioural framework. For example, a learning designer might decide that the learner could have an unlimited amount of time, but is only allowed one attempt and is never shown a solution. This would define a simple path through the diagram as *Content Bean* followed by either *Solution Correct*, or *Solution Incorrect* leading to *Out of Tries*.

A more complex possibility might be that the learner is allowed at most four tries to solve the challenge, with a five minute time out and a solution may be shown after two tries have been made. A detailed textual description of the possible paths through the framework could be made, but it should be obvious that this, and many other possible, behaviors could be afforded by the design.

The design implicitly shows all the behaviors that are possible. The relative simplicity of the design with only two superstates one with a *try/try again* pattern of behaviour and one with four possible challenge terminations suggests that it will have the required walk-up usability. Moreover the design can be used during development and testing to ensure that the framework does have the correct patterns built into it.

A particular instance of the bean will have a content bean plugged into it and be configured for a particular pattern of behavior by an instructional designer. A utility that allows the instructional designer to accomplish this will be described below after two sample learning objects are shown.

3. TWO SAMPLE CONTENT BEANS

3.1 The Imperative Programming Bean

Figure 2 shows the imperative programming content bean in the *Solution Incorrect* state configured with a time constraint, a limit the number of tries and allowing a solution to be shown.

This doesn't seem to be right!
You have 2 trie(s) left.
01:19

```
int rose = 4;
int magnolia = 9;
rose --;
magnolia ++;
magnolia ++;
```

Enter the value of rose:

Enter the value of magnolia:

Run

Try Again

Show Me

Figure 2. The Imperative Programming Bean

The imperative programming bean is concerned with affording the learner the opportunity to explore the manipulation of primitive integer variables in a C style programming language. The learner is invited to examine the code shown and predict the values of the variables after the code has executed; typing the values into the fields at the bottom of the content bean's

interface. Each time the bean is re-presented a different challenge will be given in order to assure that deep, rather than shallow, learning occurs.

Although not strictly a required characteristic of a content learning bean this, and other objects produced as proofs of the concept, support progressive disclosure. The content bean shown in Figure 2 is intended for learners who are beginning to explore imperative concepts. It is restricted to two variables, small positive integer values, three lines of code and uses only of the post-increment and post-decrement operators. The instructional designer can progressively permit more cognitive complexity by increasing the number of variables, the range of values, the number of lines of code and the operators allowed. Programming beans concerned with selection and iteration have also been developed.

3.2 The Piano Music Bean

Figure 3 shows the piano music content bean in the *Content Bean* state configured with no time constraint and to not allow a solution to be shown.

Figure 3. The Piano Music Bean

The user is required to drag and drop arrows, pointing to which keys they think have to be played for the chord specified. As with the imperative bean this bean supports progressive disclosure, allowing for ranges of complexity of chords. It can also be operated 'in reverse' where the learner has to name, using pull-down menus, a chord which is displayed.

In both of these beans, the area at the top where instruction is given and the area at the bottom where the buttons are shown are provided by the bean framework. The area in the middle is the context specific learning bean and it is the interactions in this area which comprise the *Content Bean* state on the state transition diagram. A learning object developer can, by using a defined software interface, produce a content bean for any topic provided only that it can generate challenges and decide if the learner has successfully completed the challenge set.

4. THE CONFIGURATION UTILITY

The instructional designer is responsible for preparing an instance of the bean and hosting it within a particular learning context. There are two parts to the configuration, configuration of the learning bean framework and of the content bean. The configuration utility for the bean framework is illustrated in Figure 4.

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

Figure 4. Learning bean framework configuration

Using this utility the instructional designer is able to specify the number of tries, if an answer is to be revealed, if so when the answer is revealed, if the bean is timed and if so how long is allowed. It is also possible to set the bean into non-challenge, explorative mode, and to prepare it for use in summative assessments. These latter two aspects are omitted from the design presented in section 2 for reasons of space.

Each content bean requires a specific configuration utility which can be presented to the instructional designer along with the framework utility. The configuration utility for the imperative bean is presented in Figure 5. This utility allows the complexity of the learner's task to be specified, with the range of operations chosen from the pull-down menu as shown.

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
<div> <div>Bean configuration</div> <div> POST ONLY POST-PRE ADD-SUB ADD-SUB-SELF ALL-BUT ALL-MODULO DIV-MOD ALL-BUT-SELF ALL-MOD-SELF EVERYTHING </div> </div>	
Challenge	
No. of tries	
Reveal	
Reveal limit	

Figure 5. Imperative Bean Configuration Utility

When the instructional designer has configured both the framework and the content components of the bean, an instance of the configured bean can be launched in a separate window. This allows the instructional designer to interact with the bean as if they were the learner, in order to confirm that the configuration is as they intended. The window that contains the bean instance also contains the HTML APPLET tag together with the list of parameters that define the bean and its configuration. The instructional designer can copy and paste this tag into the location within the learning environment where it is needed. As the bean is written 100% in Java and the APPLET tag is part of the HTML standard this should assure

wide deployability for the bean. The beans were also developed using java's internationalisation capability to further enhance their deployability.

5. LEARNER TRIAL

It is difficult if not impossible to differentiate the usability aspects of the framework from those of the embedded content bean in end-learner trials. Accordingly the beans were evaluated in their entirety, including both the framework and the content parts. Moreover the context for the use of these beans is pedagogic and the proof of their utility might be in the measured ability and, perhaps more importantly, confidence of the learner.

The learners involved in one trial were 17 second year undergraduate computing students who might be expected to be able and confident about their skills in the fundamental programming concepts of sequence, selection and iteration. However many studies [7] have indicated that this is rarely the case.

Table 1. End-learner trial of the programming beans

	pre test		post test	
	ability	confidence	ability	confidence
sequence	0.45	2.25	0.95	4.35
selection	0.4	3.15	0.8	4.25
iteration	0.2	1.95	0.65	3.85

The procedure consisted of a paper pre-test and post-test assessing the ability of the students to answer questions related to the concepts, scored 0, 1 or 2. This was accompanied by a five point Likart scale measuring their confidence in their ability to answer such questions. Following the pre-test each of the three beans was briefly demonstrated using a data projector.

The learners were then allowed 30 minutes to use six bean instances, two each of sequence, selection and iteration with one conceptually simple and one more complex. All beans were configured to allow an unlimited number of untimed attempts with no solution available. The students were reminded every ten minutes and asked to move onto the next pair of beans.

The intention had been that the investigator would log each request for help from the students which related to either unknown bugs or the usability of the beans, as distinct from conceptual issues relating to programming. In the event there were no requests for assistance of any kind from the students during the session and the inference was that the beans had satisfied the requirement for near walk-up usability.

The results of the trial are shown in Table 1. The ability measures are out of 2, with 2 being perfect performance. The confidence measures are out of 5, with 5 being highly confident. For all programming concepts both an improvement in ability and an improvement in confidence was demonstrated. This latter outcome is possibly more significant as confidence is a pre-requisite for effective learning [8].

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. However other in-house trials have demonstrated similar results and the beans are being widely used in several institutions attracting favourable comment from the tutors involved.

6. ACKNOWLEDGMENTS

The development of the beans was co-funded by London South Bank University and the Higher Education Academy subject centre for Information and Computer Sciences. The development of the beans was done by Gus Morataro under the direction of Fintan Culwin. This paper has a 8% and 12% overlap with two previous papers on learning beans [4,9] which can be attributed to a common technical vocabulary. The beans are available free of charge for non-commercial use at <http://cise.lsbu.ac.uk/pooples>.

7. CONCLUSIONS

This paper has illustrated how a highly interactive behavioral framework can be designed by use of state transition notation and subsequently implemented. Two content beans were presented, provided proof of concept evidence that the separation of the generic pedagogic behavior from the learning topic specific behavior can be effected. In the wider context the beans are configured and deployed by an instructional designer and a configuration utility to assist in this was also described. Trials of the beans with learners indicated that the beans had accomplished their design requirement of walk-up usability and providing an effective and engaging learning environment.

8. REFERENCES

- [1] McGreal R, *Learning Objects: A Practical Definition, an attempt to bring some sense to the definition of a LO*. International Journal of Instructional Technology & Distance Learning, 2004.
- [2] Gabriel, Richard P., *Objects have failed. Notes for the OOPSLA 2002 panel discussion*. (Available at <http://www.dreamsongs.com/NewFiles/ObjectsHaveFailed.pfd> Accessed 25 June 2007), 2002.
- [3] Sun, *Enterprise Java Bean White Paper*. (Available at <http://java.sun.com/products/javabeans/docs/spec.html> Accessed 25 June 2007), undated.
- [4] Culwin F. *Beyond learning objects: Towards learning beans*. In R. Atkinson, C. McBeath et al (Eds), *Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference*, 2004.
- [5] Dalziel J *Implementing Learning Design: The Learning Activity Management System (LAMS)*. In G.Crisp, et al (Eds), *Interact, Integrate, Impact: Proceedings of the 20th ASCILITE Conference Adelaide*, 2003.
- [6] Culwin F. *The statechart design of a novel date input mechanism*. *Italics* V3 No 1 Jan 2004. Available at <http://www.ics.heacademy.ac.uk/italics> (Accessed 25 June 2007), 2004.
- [7] McCracken M. et. al, *A multi-national, multi-institutional study of assessment of programming skills of first year CS students*. *ACM SIGCSE Bulletin* Vol 33 Issue 4, 2001.
- [8] Norman, M. and Hyland, L., *The Role of Confidence in Lifelong Learning*. *Educational Studies*, Vol 29, No 2/3, 2003.
- [9] Morataro G & Culwin F *Three Programming Beans*, *Proc. 7th Annual HEA/ICS Conference Dublin*, 2006.