Internet Distance Education: Object-oriented Modeling in Instructional Design

Charles Boulet
Internet Distance Education: Object-oriented Modeling in Instructional Design

Rights
Terms of use for work posted in CommonKnowledge.

This article is available at CommonKnowledge: http://commons.pacificu.edu/inter08/4
Internet Distance Education: Object-oriented Modeling in Instructional Design

by Charles Boulet

Preface

This discussion centers on Distance Education (DE) in general and introduces Object-oriented Distance Education modeling (OODE), and remains largely neutral with respect to development or deployment platforms. Drawing examples from the grade-school classroom to large industrial and government applications, the goal is to establish general guidelines for approaching instructional design for online delivery models based on a synergy of well-established technical and pedagogical constructs. There are many ways through which DE can be delivered and managed, but the final technological implementation is a secondary concern and will only be considered tangentially.

For purposes of this discussion, Distance Education is defined as Internet-mediated instruction where the learner is not physically be in the presence of the instructor and assumes varied degrees of synchrony in interactions between the learner and the provider. The discussion recognizes that any solution for Distance Education can equally well be implemented on a local level to enhance or run in parallel more traditional models of delivery. Furthermore, given the wide array of instructional purposes and delivery limitations, it is impossible to do more than simply introduce and illustrate concepts. Please consult the references for further detail.

Introduction

In the previous article, it was noted that educators had learned some important lessons in implementing computer technology in the classroom and school division. As military and government bodies had known for decades before schools, computer technology and networking
can facilitate enormous growth in research and development, and new and extended efficiencies internally to the organization and in dealing with outside parties, organizations, and clients. From the mid-1980s to 2000, school divisions for their part learned that computer workstations and data storage lend themselves particularly well to automation of routine reporting and data collection, as notable but not solitary examples, but they are poor substitutes in the classroom. From the perspective of curriculum development, educators learned that it is best to learn to use computers to accomplish relevant and necessary tasks rather than to understand how and why they work.

Today, educators have a unique opportunity in history, pioneering the universe of Internet-mediated Distance Education. In the last ten years, there has been explosive growth in web-based services and technologies yielding new services for data management and reporting, to enrollment, curriculum, and support gateways. Evolving in parallel, third-party contractors have proven themselves to be worthy and capable of technical provision, design, and management, all of which is often poorly managed and under-funded in educational institutions and providers. All this leaves the door open to focus on new and effective ways of doing what educators do — teach.

The current state of technology implementation in education reflects a trend towards a more reasonable and balanced integration of skills and expertise with educators using technology to enhance and extend delivery, allowing new tools to facilitate the process rather than confound it. In this article, we will continue the discussion by focusing on the modeling of instructional constructs for Distance Education from the dual perspective of technology and pedagogy, the former from the view of Object-oriented constructs, and the latter from the view of Bloom's taxonomy for learning, teaching, and assessment. Several benefits of DE will be exposed through a discussion of how formalized well-structured instruction enhances the experience and production of online education.
Contrasting Today and Tradition

More traditional approaches to teaching, by their nature, revolve around the instructor. A student must be delivered to some center of learning, entering as a guest under several conditions, and becomes the pupil, a necessarily subordinate role. The student waits upon the instructor and is completely dependent upon the instructor, and to a lesser extent his fellow students, for learning. These factors alone impose serious restrictions on the learner as he is forced to internalize new rules of conduct and adapt to a new environment and instructional style before he can fully avail himself of the opportunity to learn. Even then, the learner is further limited by the abilities of the instructor to assess his learning style and level of comprehension then adapt the instruction accordingly. Furthermore, the student runs the risk of having an instructor who, for innumerable possible causes, cannot effectively execute the task or might subject the student to unfair treatment or bias. Whereas in the ideal state, the traditional approach to instruction allows for a guided, personalized and socratic solution to instructional needs, the reality is that the modern classroom, being overly dependent on the instructor for delivery, falls short on the promise. This is not to say that good teaching does not occur, teachers do not always have the time nor skills to do the job effectively. They remain, however, the best source for tailored and meaningful instruction and evaluation compared to any automated system that might exist.

In its simplest form, Internet-mediated Distance Education offers many options for pre-packaged rectilinear asynchronous training solutions for a variety of needs. The student typically will pay a fee for access to video content and will need to connect to the provider's site in order to gain access to it. This approach typically offers text, audio, and video content which the user works through in a set sequence. If any questions arise with respect to content, the user can play back the audio and video or re-read selected passages but only rarely has access to experts to consult. There are often exercises or sample files the student works with, most often by mimicking what the recorded instructor does and there is
usually no facility for feedback on performance or troubleshooting help. This format is particularly effective in technical skills training for desktop computer productivity tools such as word and image processing, web and paper publishing, and others where instruction requires only that students be exposed to new procedures limited scope or more complex procedures consisting of simpler tasks chained in a predictable sequence. While this approach allows for a simple retail 'on-demand' approach to training, it does not address concerns that arise from a need for more interactive instruction and evaluation required by more complex learning situations. It does, however, provide a great deal of flexibility for the student as they can approach learning at a self-determined pace and schedule. The student is further freed from the constraints of space, instructor bias, and other psychosocial factors present in a more traditional setting.

Today's web provides advanced facilities to more flexibly approach individual learning styles and so the simple pre-packaged solutions described above are not representative of what is possible. Indeed, new DE paradigms are making use of advanced algorithms and summative and formative assessment in order to more closely approximate training solutions to students' specific needs. In addition, the combined use of varied synchronous and asynchronous communication modalities allows for human interaction and professional intervention where required or desired. Whereas facilities for managing and delivering data to the student continue to evolve more robust and flexible solutions for DE, the foundation of effective teaching and learning remains in sound instructional design. DE can then be implemented in such a fashion as to meet the requirements of instruction, rather than altering instruction to meet limits of technology.

**Designing for DE**

Does the mode of delivery impact on instructional design? Should it? In practical terms, the answer to both questions is yes; the design of an instructional solution depends on myriad factors including but not limited to budget, availability of existing solutions and components,
staffing, target audience, numbers of pupils versus instructors, availability of broadband and print materials, mail service, physical limitations of students and instructors. In principle, however, the answer to both questions is no. Effective instruction means learning is facilitated and guided through key milestones leading to the final outcome, whatever that might be, regardless of how interactions between student and instructor are mediated.

The primary functional distinction between traditional and DE delivery models is the role played by the instructor. The classroom teacher, provided she knows the subject matter well enough, could begin teaching with little to no advance preparation, correcting herself as she moves through curricular goals; she becomes the center of the process and the learning is entirely dependent on her performance. DE, in contrast, forces the instructor to abstract her lesson, to deconstruct the goals and objectives of curriculum and very deliberately build lessons according to known rules and predictable processes; the student assumes more of a prominent central role while the instructor, though present, is not the focus.

The process of designing by object-oriented principles for DE, as presented herein, illustrates several benefits of adopting more of a formal structure to instructional design. In particular, by borrowing constructs from relational database management and object-oriented programming and combining them with well-established pedagogical principles, instructors can create learning solutions that can enhance flexibility and efficacy of DE from the view of programming and instruction, with similar positive effects when implemented in conjunction with more traditional instruction.

Designing instruction from a detached perspective allows the instructor to appreciate what is important and to lose what is irrelevant and potentially confounding. Furthermore, following more formalized principles of abstraction and atomicity leads to simpler yet pedagogically comprehensive instructional models, which are more easily adapted to varied learning styles. Students will appreciate and are
protected by the clarity, predictability, and lack of cultural and personal bias in well-abstracted models. Additionally, thoughtful and formalized instruction protects the instructor from grievances and frees her to concentrate more on instruction.

From an administrative perspective, such modeling provides maximum efficiency as each component defined is required and only ever created once, as a class, for example. The full course offering of the organization becomes a catalogue from which customizable solutions may be drawn. These classes may be combined in a variety of digital media from desktop applications to web-based delivery modes to meet the needs of DE and locally situated students.

Paradoxically, of the many great benefits of approaching development in this way, perhaps the most significant advantages relate to neither pedagogy nor technology in a direct way. First of all, adaptation to this sort of design methodology only requires that the instructor adds more detailed form to his instruction and removes himself from the process. In this way, the problem of personality is eliminated. For the instructor, this simplifies management of students, and the organization gains in terms of transitional problems that occur with changing staff and curriculum. Secondly, OODE models provide maximum technological accessibility and flexibility for all stakeholders from students to managers and directors. This promotes productivity and creates new opportunities for reaching new markets.

**Data Normalization and Atomicity**

Let us begin the discussion by considering briefly a relational database concept, that of Normal Forms (NF). Databases, in the most general terms, contain tables of data which are referenced alone or in combination in order to add or retrieve simple and complex values. Database tables are defined by columns first, then rows. Columns define what the values represent and what constraints might be applicable. For example, a table might have columns for LastName and FirstName defined as a text field of maximum length 50 characters each. The same
A table might also have a column defined as a date field called BirthDay. By comparison, each row of the table would include unique values for each of the described columns; our table would contain numerous rows, each one representing a single person.

First elaborated by Edgar F. Codd, the Normal Forms provide criteria for designing database tables in such a way that risks of data compromise and inconsistencies are limited. Normalizing tables often also has the corollary effect of rendering data stores more efficient and flexible. Let's consider briefly the core principles of the first three (of several) Normal Forms for tables. (This is not intended to be a detailed presentation of Normal Forms, nor the controversies that might surround their definitions.)

First Normal Form (1NF):
Eliminate duplicate columns from the table (in other words, eliminate redundancy) and
Create separate tables for each group of intrinsically related data, and
Identify each row with a unique column (a key).

Second Normal Form (2NF)
Meet all the requirements of the first normal form, and
Remove subsets of data that apply to multiple rows of a table and place them in separate tables, and
Create relationships between these new tables and their predecessors through the use of foreign keys.

Third Normal Form (3NF)
Meet all the requirements of the second normal form, and
Remove columns that are not dependent upon the primary key.

In 1NF, we gather data into related groups and identify each group (table) with an identifier or key. The key identifies the row and has no other value to the dataset. In 2NF, we verify that all rows in the table represent the same constructs - a table of people, for example, should not include the name of a commercial enterprise. Using keys, we create
relationships between the separated tables (TablePeople and TableEnterprises) so that, for example, John Smith's key is associated with a row in TableEnterprises (Smith and Co. Ltd.). 3NF takes another look at the tables and encourages us to eliminate rows that are not intrinsically dependent on the primary of the table. So, for TablePeople, each row would be identified by a primary key (a.k.a. the 'key' or unique identifier) and would contain information specific to the construct of 'person'. A column in the TablePeople for bank accounts would not make sense, according to 3NF, because the TablePeople key defines the person as opposed the banks they use. Another separate table, TableAccounts, would be better suited to contain this information as 'Bank Account' is a construct completely distinct from 'Person'.

**Class-based Design**

The second computing science concept called into play is that of object-oriented design. Object-oriented programming (OOP), first introduced in the 1960s through Simula 67 then formalized and expanded later by IBM as Smalltalk in the early 1970s and others since, allows programmers to model the real world by creating classes of objects, each of which has its own internal workings, and defines its own data and behavior. Classes are used as a type of flexible template for the creation of course and lesson objects. Given the predictable and well-structured definition of the classes and objects, they may also be recombined as building blocks of new courses and lessons.

These classes, and the objects created from them, relate to one another through exposed interfaces and otherwise hide the details of their operations, in other words, they are self-contained. As a metaphor, you might ask a car salesman to accept a certain price on a vehicle and he will give you the answer without divulging the details of how he worked out their response, but he might convey some of the hidden information to a trusted partner, such as the dealership's business manager. To further the example, the salesman will understand a certain set of data constructs (such as English words), but will not know how to manage others (such as Finnish). This is to say that classes within an
organization will share common interfaces with constructs of similar names behaving in similar fashions from department to department.

The following are key concepts in OOP as they relate to DE design:

**Class** - A class lists the traits of a thing (object), in other words it 'abstracts' the thing - in this case students, instructors, courses and lessons. The class includes the thing's characteristics (its attributes, fields or properties) and the thing's behaviors (the things it can do, or methods). For example, the class ClassStudent would consist of traits shared by all students, such as firstName, lastName, mailingAddress, emailAddress, and enrollmentStatus (characteristics or 'properties'), and the ability to enrollNewClass, dropExistingClass, makePayment, takeTest (behaviours or 'methods'). Classes provide modularity and structure in an object-oriented computer program. The inner workings of a class (code to a programmer, instructions to an educator) should be relatively self-contained, in other words encapsulated. Collectively, the properties and methods defined by a class are called members.

**Object** - A particular instance of a class. Whereas 'ClassStudent' is a generalized construct, 'Sally_Peterson' is an instance of the class, with specific properties defined. We say the object Sally_Peterson is an instance of the ClassStudent class. The set of values of the attributes of a particular object is called its state. The object consists of a particular state and the behaviours defined in the object's class.

**Method and Properties** - As described earlier, methods are an object's abilities, in other words, the things it is allowed to do and designed to accomplish. enrollNewClass, dropExistingClass, makePayment, takeTest are all examples of methods defined for the ClassStudent and its instances (the objects created, or instantiated, from it and the subclasses derived from it). Obviously, it makes no sense to design and assign a method (ability) to a student that the student would or could never use, though students might have methods defined for them that they are conditionally allowed to execute (perform). Properties are simply characteristics, or traits, of the objects. Examples of properties of
the ClassLesson class might be NumberOfCredits (3 or 4 or whatever), Available (yes or no), and Enrollment (being the number of students enrolled currently).

**Inheritance** - 'Subclasses' are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own. For example, the class ClassStudent might service as templates for the sub-classes called ClassGraduateStudent, ClassUndergraduateStudent, ClassMedicalStudent, or ClassLawStudent. Clearly, all of these are students, but they have important differences. The subclasses can inherit properties and behaviours but can also add their own tailored members. ClassMedicalStudent will inherit the basis information required for all students and may add properties and methods to describe medical specialty training and immunization status, but this information would not be required for undergraduates, for example. Finally, classes may draw traits from not only their parents but also from other classes with which they have permission to communicate.

**Encapsulation** - Encapsulation conceals the functional details of the inner workings of the class from objects that communicate with it. In terms of DE design, encapsulation requires that there be no guessing as to how the lesson be carried out. The lesson may reference antecedent work inherited through its parent class (that is, all students must have some core set of properties and methods) and so it does not always have to redefine them, but anything else that has not been inherited must be deliberately and explicitly defined. In practical terms, the student should not have to guess as to requirements, instructions, or their status.

**Abstraction** - Abstraction is simplifying complex reality by modeling classes appropriate to the problem, and determining which of the parent class members to inherit to the new child. Abstraction of the elements of a course seems intuitively simple but can be difficult, even more so for a particular lesson, because it demands that we define many things we typically take for granted, have forgotten, or have simply integrated into
our daily lives as instructors. The instructor must adopt the outside view, that of the student newly approaching the learning environment.

**Polymorphism** - Polymorphism allows you to treat derived class members just like their parent class's members. More precisely, polymorphism allows the programmer/designer to have the object (an instance of a class) respond to the same method call in accordance with those behaviours defined for them. In other words, the same method will be handled differently depending on who is carrying out the action. As an example, ClassGraduateStudent and ClassUndergraduateStudent would both have the method/ability to enroll in a course, call it the enrollCourse method. Both students would make the same request (call the method), but it might be handled differently from an administrative perspective. The point is that simplicity arises when the interface relies on predictable methods and procedures so that whichever student you are dealing with has the same command set. Flexibility arises because even though the same action is requested by different classes of students, the request is handled according to the rules defined for that particular class of student. The point here is not to 'program' courses and lessons as though they were computer programs, but rather to use these concepts as guides in the deconstruction of learning needs and subsequent reconstruction of more effective and flexible models of instruction. Furthermore, not every lesson need reflect all of the principles presented in order to be effective.

**Fusion of Technology and Pedagogical Foundations**

These principles of deconstruction and reconstruction, when combined with the learning paradigms described by Bloom et al. (1956) and Anderson & Krathwohl (2001), or by Guilford (1956, 1966, 1967) and Meeker (1969), the structures that result can yield formidable and efficient instructional models, which can easily adapt to DE and traditional models.

To review, Bloom's Taxonomy, or simply 'The Taxonomy', elaborates the cognitive process dimensions of learning in terms of six types, each
of which is further sub-divided into measurable behaviours. This represents the Cognitive Process Dimension:

Remember: Recognize, Recall
Understand: Interpret, Exemplify, Classify, Summarize, Infer, Compare, Explain
Apply: Execute, Implement
Analyze: Differentiate, Organize, Attribute
Evaluate: Check, Critique
Create: Generate, Plan, Produce

Each of these juxtaposed against the axis of the Knowledge Dimension consisting of Factual Knowledge Conceptual Knowledge Procedural Knowledge Meta-cognitive Knowledge
The resulting arrangement appears thus: **Remember Understand Apply Analyze Evaluate Create** Factual Conceptual Procedural Meta-cognitive

On the simplest level, these axes can be accounted for intuitively in the design of the lesson classes. Put another way, the instructor can 'wing it' and assume that she is covering the wide range of cognitive skills required by the program of studies. The preferred approach is to incorporate the skills in an array in the lesson classes (hence objects) themselves, the table indicating which elements have been taken into account in the particular lesson. In this view, the skills are thus represented as properties of the class available to be referenced externally to other instructors who might want to make use of the lesson object in question. Further, the object taxonomy properties for an entire course could be queried and assessed in order to determine the nature of the cognitive skills and knowledge inherent in the program of studies.
**Approaching Instructional Design for DE:**

To achieve the best results in instructional design for DE, the same guidelines apply, regardless of the nature of the learning task. As a first priority, use well established models of learning to clearly conceptualize specific learning outcomes incorporating the requisite knowledge elements. The next step is to then follow the principles of relational databases and object-oriented programming to guide the deconstruction of the learning requirements leading to the reconstruction of the instructional model and lessons.

Consider what knowledge, skills, attitudes and, as appropriate, affective responses are desired as relevant outcomes.

Sequence activities and course requirements in such a way that students can achieve progressive successes in prerequisite skills as they move forward in their learning.

Analyze activities individually ensuring all prescribed cognitive, affective, and behavioral (skill) requirements have been incorporated.

Plan out content as a function of outcomes bearing in mind the specific learning dimensions addressed in the lesson. Ensure that the curriculum and content is addressing not only the knowledge but the cognitive skills required by the program.

Eliminate redundancies in the course and its lessons, unless specifically required for enhancement and reinforcement of learning.

Clearly define pre-requisites and post-requisites and other external dependencies.

If the same outcomes are covered in another lesson or in another course, consider using that lesson in the current instance.
If the information is available elsewhere, reference it instead of creating a new instance of the data.

Eliminate unnecessary dependencies between elements. Is the instructional sequence the way it is because it makes sense, or rather because of tradition? Can the sequence be changed without impacting achievement of global objectives?

Think in terms of Classes — Identify first the levels of abstraction required, perhaps the highest level is the 'degree' or 'certificate', or it may simply be 'course'. Each course is generally then divided into lessons. Each lessons contains learning and evaluation elements. Clearly define the methods (activities allowed the students and instructors) and properties (resources, statistics, other state elements) required for the highest classes and then inherit the relevant members to the subclasses, making members more specific according to the needs of the subclass.

Consider the Object — Let the nature of the final object guide the design of the class. Consider, for example, not 'Sally Peterson' so much as the class she represents. What makes her different from others in her group? If the answer is that there is nothing relevant that separates her from her peers, then they are all in the appropriate class. Define then all the relevant common terms between Sally and her peers including the things she needs to do, or have done for her (the methods), and the information required to manage Sally as a student (properties, state).

Methods and Properties — Avoid the specificity trap when dealing with class and class member definitions. Approach the problem from the view of determining those members which absolutely must be defined, leaving everything else to be ignored.

Inheritance — The highest level of abstraction should be determined by a central governing body in an organization and inherited downward to departments and classrooms. By extension, all departments inheriting from the same parent would be able to freely interface with the others, making full use of available libraries from other areas.
Encapsulation — Instructions and procedures should be unequivocal and consistent throughout the entire program (degree or course or lesson). Be sure to indicate where members are polymorphic with respect to parent classes, but this need only be noted within the child class itself as the specifics are irrelevant to other classes.

**Conclusion**

This paper presents an outline to an approach to instructional design that borrows a conceptual framework from computing science and merges it with well-established pedagogical principles in order to establish more efficient and flexible models of delivery. Relational database normalization provides a paradigm for atomizing data and rendering it more useful to varied aspects of the training organization. Concepts drawn from object-oriented programming further allow the designer to abstract learning and instructional constructs in such a fashion as to render it more useful to the organization and more easily comprehensible from the perspective of students and instructors.

The next paper will present a generalized example of instructional design following OODE principles. While OODE lends itself well to delivery in the context of digital media, it will be shown that the same principles will work equally well when implemented locally in more traditional settings.

**References:**


Meeker, Mary N., The Structure of Intellect: Its Interpretation and Uses.


The International Review of Research in Open and Distance Learning - http://www.irrodl.org/index.php/irrodl/index


For a comprehensive listing of instructional design (ID) models, see http://carbon.cudenver.edu/~mryder/itc/idmodels.html