MOOClets: A Framework for Dynamic Experimentation and Personalization

Joseph Jay Williams
Harvard University
Cambridge, MA, USA
joseph_jay_williams@harvard.edu

Anna N. Rafferty
Carleton College
Northfield, MN, USA
arafferty@carleton.edu

Samuel Maldonado
San Jose State University
San Jose, CA, USA
samuel.maldonado@sjsu.edu

Andrew Ang
Harvard University
Cambridge, MA, USA
andrew_ang@harvard.edu

Dustin Tingley
Harvard University
Cambridge, MA, USA
dtingley@gov.harvard.edu

Juho Kim
KAIST
Daejeon, South Korea
juhokim@cs.kaist.ac.kr

ABSTRACT

Randomized experiments in online educational environments are ubiquitous as a scientific method for investigating learning and motivation, but they rarely improve educational resources and produce practical benefits for learners. We suggest that tools for experimentally comparing resources are designed primarily through the lens of experiments as a scientific methodology, and therefore miss a tremendous opportunity for online experiments to serve as engines for dynamic improvement and personalization. We present the MOOClet requirements specification to guide the implementation of software tools for experiments to ensure that whenever alternative versions of a resource can be experimentally compared (by randomly assigning versions), the resource can also be dynamically improved (by changing which versions are presented), and personalized (by presenting different versions to different people). The MOOClet specification was used to implement DEXPER, a proof-of-concept web service backend that enables dynamic experimentation and personalization of resources embedded in frontend educational platforms. We describe three use cases of MOOClets for dynamic experimentation and personalization of motivational emails, explanations, and problems.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; K.3.1. Computer Uses in Education

Author Keywords

A/B experiment; dynamic experimentation; MOOClet; personalization; multi-armed bandit; reinforcement learning; statistical machine learning; adaptive learning

L@S 2017, April 20 - 21, 2017, Cambridge, MA, USA

ISBN 978-1-4503-4450-0/17/04...$15.00

DOI: http://dx.doi.org/10.1145/3051457.3054006

INTRODUCTION

One of the promises of online education is to advance research on learning, and randomized experiments in online educational environments have yielded many insights into learning and motivation. However, it is important for academic research to yield not only generalizable principles, but result in enhancements to student learning. Unfortunately, there is a substantial delay and many obstacles between obtaining experimental results to making concrete changes to courses. Arguably, most published randomized experiments in online environments do not directly result in improvements to the context in which the study was conducted.

In contrast, when randomized experiments or "A/B tests" are used for product testing versions of websites and online advertisements, the software is designed so that versions that maximize user engagement or purchases are provided to subsequent users [3]. Instructors and students could benefit more directly from research studies if software for educational experiments was similarly designed to use data to dynamically transition from assigning versions of a lesson with equal probability to assigning effective versions more frequently for future students, rather than using data from an experiment in one course to impact design in a course occurring months later.

However, using data from randomized experiments to provide one version of a resource to everyone reflects a "one-size-fits-all" assumption, and could miss what are known as heterogeneous effects, where one version works well for one subgroup of learners, while another version works better for a different subgroup. Experiments could therefore lead to personalization, in the sense that different conditions are presented to learners with different characteristics. While personalization encompasses many different approaches, such as intelligent tutoring systems adapting how many activities are provided or customization of choices by individual learners (see [2] for a review), adaptive assignment to different versions of software suggests one way to bridge experiments with the large body of work on adaptive learning and personalization.

Despite the novel opportunities online environments provide for randomized experiments and their widespread use, relatively little work has examined how to design tools for instructors and researchers to conduct experiments. The major technical work on building tools for experiments has taken place in industry settings like website testing, where companies like Facebook and Microsoft invested resources to implement experiments and machine learning algorithms for product improvement, like creating programming languages for experiments [1]. This paper addresses the need for a more standardized way of conducting experiments in educational resources by proposing a software requirements specification that can be reused in different contexts and eases the burdens of customizing experiments through an API that facilitates experimentation, dynamic improvement, and personalization. We describe three case studies that used different implementations of this specification within learning resources.

DESIGNING SOFTWARE FOR EXPERIMENTATION, DYNAMIC IMPROVEMENT, AND PERSONALIZATION

We identify three major functions to support in the experimental specifications, which are closely related to each other: (1) Conducting an experiment on a resource, by assigning alternative versions of the resource with equal probability. (2) Dynamically improving a resource, by adding or removing which versions are presented over time, as new ideas arise or new data is collected. (3) Personalizing a resource, by presenting different versions to learners based on characteristics like prior knowledge or motivation.

Design Goals

Tools for conducting an online educational experiment can simultaneously enable dynamic improvement and personalization, if it is possible to:

- Add and remove versions at any point in time.
- Use multiple methods for deciding how versions are delivered to a particular learner, including but not limited to uniform randomization, weighted randomization, personalization based on a learner's characteristics.
- Change the method for assigning versions to learners, at any point in time, such as changing the weights/probabilities for assigning a version, changing the rules for assigning based on a learner's characteristics, or both.
- Collect and access data from past learners who received alternative versions, in deciding which versions are best and in dynamically improving a resource.
- Collect and use data about a specific learner, in deciding which version to assign to them for personalization.

MOOClet Specification for Software for Experimentation, Dynamic Improvement, and Personalization

To achieve the preceding design goals, we provide what is known as a *software requirements specification* for what components and functions should be satisfied by the software underlying an experiment on a digital educational resource. We name this the *MOOClet* requirements specification, and the

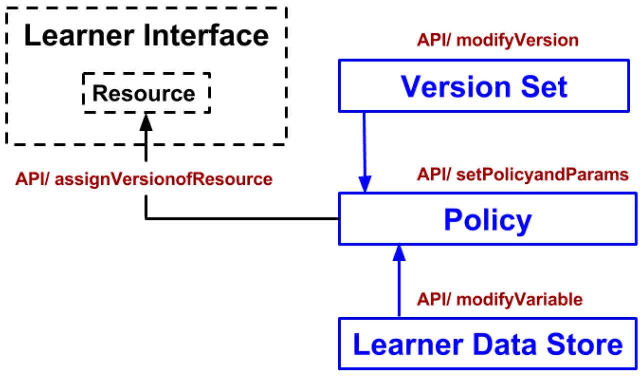


Figure 1. Components of the web service that enable an educational resource to function as a MOOClet. Before a learner interacts with the resource, the Learner Interface makes an API call to the web service, and the Policy associated with the MOOClet can use information from the Learner Data Store in selecting a version from the Version Set and serving it to the Learner Interface. API endpoints allow addition or modification of versions and variables, and changes to the current Policy.

API (Application Programming Interface) Specification		
Endpoint	Parameters	Function
assignVersionofResource	learner_id, mooclet_id	Assign version of resource using current policy
modifyVersion	version_id, mooclet_id, version_content	Add or modify a version for a MOOClet
modifyVariable	learner_id, mooclet_id, variable, value	Add or modify variable in Learner Data Store
setPolicyandParams	mooclet_id, policy, policy_parameters	Change policy and/or policy parameters

Figure 2. The key API endpoints for the DEXPER web service. DEXPER serves as the backend for MOOClets by providing versions to a frontend Learner Interface, and allowing modification at any point via API of Versions, Learner Variables, and a Policy or its parameters.

term *MOOClet*[1] is used to refer to any educational resource augmented to meet these requirements. Roughly, implementing a digital educational resource as a MOOClet associates the frontend software presenting the resource to learners with the backend software that maintains a set of versions, a store of learner data, and a suite of methods for assigning versions to learners (including rules for weighted randomization and personalization). A MOOClet must also provide functions that instructors or researchers can use at any time to modify the backend components, such as adding new versions or learner data, and changing the method for assigning versions to learners.

As depicted in Figure 1, an online educational resource implemented as a MOOClet – or more formally, using the software requirements specification for a MOOClet– has the following components:

- A *Learner Interface*, which displays the content a learner interacts with. The exact content the learner sees depends on which alternative condition or version of an educational resource they are assigned to. For example, one of two explanations for why the answer to a problem is correct. This is what would typically be called an educational resource,

[1]We introduce the novel term *MOOClet* because it is useful to have a label for educational resources that matches this precise specification, versus educational resources that only enable randomized experimentation, or only enable personalization. The approach applies to any digital educational resource, and to digital resources beyond education, from websites to emails to mobile apps. Another term we have used is *AdapComp*, short for Adaptive Component.

since there is usually just one version when no experiment is being conducted.

- A *Version Set*, which contains alternative versions of content that will be provided in the Learner Interface. For example, this structure could contain multiple different explanations a learner could be shown. Elements of the Version Set can be accessed, modified, and added at any point.
- A *Learner Data Store*, which contains a set of variables linked to each learner by their anonymous learner ID. This might include whether a learner had gotten a previous problem right, or their rating of an explanation. Variables can be added and modified at any point. All variables can be accessed and used by the *Policy* that determines which version is assigned to a particular learner.
- A *Policy*, which is a function for determining which version of a resource is presented to a particular learner, and can use any data in the *Learner Data Store*, about the current learner or the effects of versions on past learners. A MOOClet must always have access to the *Randomized Personalization* Policy Class, which enables uniform randomization, weighted randomization, and personalization using IF-THEN rules. Particular MOOClet implementations can optionally provide a range of other Policy Classes.

DEXPER Web Service: Enabling Resources to Function as MOOClets

We used the MOOClet specification to implement a proof-of-concept web service that enables cross-platform dynamic experimentation and personalization, which we call DEXPER. DEXPER can be used with text and HTML components of educational resources, which are ubiquitous across platforms and play an important role in learning, such as motivational messages, explanations, and problems or quizzes.

DEXPER is implemented using Django (a Python-based framework for web applications), with classes that allow the creation and modification of SQL database objects for text and HTML versions that are needed for a MOOClet's *Version Set*, variables and values that make up the *Learner Data Store*, and a range of Policy Classes. These components of the MOOClet can all be added or modified using a set of RESTful API calls, a specification of which are shown in Figure 2. We also use built-in Django capacity to provide a graphical user interface that allows manual editing of these data objects.

The DEXPER Policy Classes allow weighted random assignments (e.g., from [0.50, 0.50] to [0.20, 0.80] to [0.0, 1.0]). The weightings can be dependent on characteristics of the learner to combine experimentation and personalization (as shown in Figure 3). It also includes a policy for dynamic improvement using Thompson sampling, a multi-armed bandit algorithm from reinforcement learning [4]. In this policy, a target outcome variable from the Learner Data Store is identified, and the algorithm chooses condition assignments to maximize this value of this variable across learners. At any point an API call can be used to change the Policy being used for a specific MOOClet (or equivalently, to change the Policy's parameters). Figure 3 shows the use of specific Policies in the use cases discussed in the next section.

MOOClets provide an abstraction for reinforcement learning

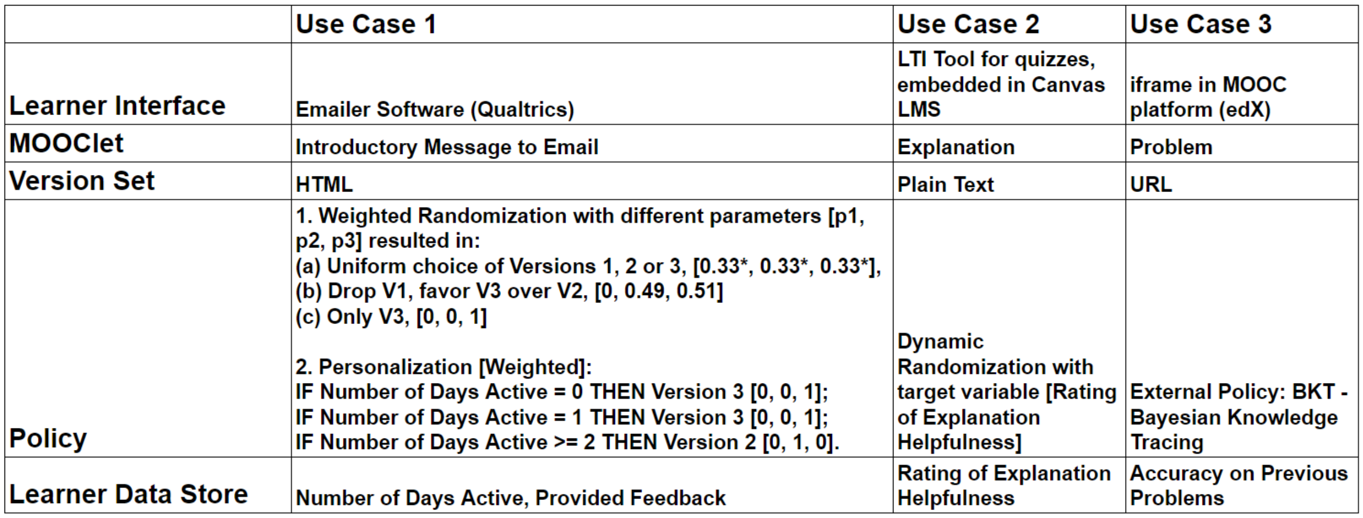
More generally, the MOOClet specification serves as an abstraction for any reinforcement learning algorithm to provide the Policy for a MOOClet. To use the terminology of RL, Actions correspond to Versions, a State Space is implicitly defined by learner characteristics in the Learner Data Store, and Reward functions are based on data in the Learner Data Store about past learners' responses to versions. To provide extensibility so that any machine learning researcher or service can provide policies for a MOOClet/experiment, DEXPER provides a Policy that functions as a pass-through: it sends data from the Learner Data Store to another API or code base to obtain a recommendation, and then allows the external method to specify which version DEXPER selects and sends to the Learner Interface. This allows researchers to take advantage of existing implementations of algorithms without needing to design their own interface between the algorithm and the educational resource.

APPLICATIONS OF MOOCLETS

This section describes three educational resources that were implemented as MOOClets, which illustrate the use of the requirements specification and DEXPER.

Use Case 1: Personalizing Emails. Learners in a MOOC were sent emails that elicited feedback about why they were not continuing with a course [4]. Different introductory messages to the emails were written, with the goal of maximizing the number of people providing feedback. The *Version Set* consisted of three versions of email content (conditions in experiment) differed in the introductory line, labeled Survey, Acknowledgement, Brief. The survey/emailer software Qualtrics functioned as the Learner Interface, and was used to send emails and provide the survey to collect learners' feedback. The Number of Days Active for each learner was obtained from the MOOC provider and sent via API to the Learner Data Store, in which all learners were identified by an anonymous ID (that could be connected by us to their email address). The outcome variable, Provided Feedback, was whether a learner responded to an email within one week. The value of this variable was added to the Learner Data Store every time a participant finished a survey, via an API call from the survey software Qualtrics.

The first batch of 1883 learners were assigned to emails with equal probability, using the Policy Weighted Randomization with parameters [0.33, 0.33, 0.33*]. The second batch of 1882 learners were assigned to emails using two policies. The first policy was to "roll out the best version" by using `WeightedRandomization`[0, 0, 1] to assign only the version with the highest response rate from the first batch of learners, the Survey message. This produced a response rate of 10.4%. The second policy was to personalize to subgroups of learners, since analyzing data from the first batch revealed a "crossover" effect. Learners with high course activity were most likely to respond to the highest rate to the Acknowledgement message, but learners with lower course activity were most likely to respond to the Survey message. Using this *Personalization* Policy resulted in 11.2% of learners providing feedback. This increase of 0.8 in response rate corresponded to a 7.6% ad-



	Use Case 1	Use Case 2	Use Case 3
Learner Interface	Emailer Software (Qualtrics)	LTI Tool for quizzes, embedded in Canvas LMS	iframe in MOOC platform (edX)
MOOClet	Introductory Message to Email	Explanation	Problem
Version Set	HTML	Plain Text	URL
Policy	1. Weighted Randomization with different parameters [p1, p2, p3] resulted in: (a) Uniform choice of Versions 1, 2 or 3, [0.33*, 0.33*, 0.33*], (b) Drop V1, favor V3 over V2, [0, 0.49, 0.51] (c) Only V3, [0, 0, 1] 2. Personalization [Weighted]: IF Number of Days Active = 0 THEN Version 3 [0, 0, 1]; IF Number of Days Active = 1 THEN Version 3 [0, 0, 1]; IF Number of Days Active >= 2 THEN Version 2 [0, 1, 0].	Dynamic Randomization with target variable [Rating of Explanation Helpfulness]	External Policy: BKT - Bayesian Knowledge Tracing
Learner Data Store	Number of Days Active, Provided Feedback	Rating of Explanation Helpfulness	Accuracy on Previous Problems

Figure 3. Overview of three uses cases for a MOOClet, showing what played the role of Learner Interface, Version Set, Policy, and Learner Data Store.

vantage of personalization over choosing the apparently "best" condition.

Use Case 2: Dynamic Explanations. To provide learners with explanations of why the answer to a math problem is correct, the AXIS system [5] automatically enhances the explanations learners received by crowdsourcing them from learners, presenting them to future learners, and dynamically choosing the most highly rated. Anyone can create systems like this using DEXPER and MOOClets. We implemented the displayed explanation as the *Learner Interface* to a MOOClet. The *Version Set* contained different versions of explanations. The variables added to the *Learner Data Store* included: the version/explanation assigned to each learner, and each learner's rating of the explanation they received. The *Policy* was DEXPER's `DynamicRandomization`, with the variable to optimize being learners' rating of a version of an explanation.

Use Case 3: Problem Recommendation. We used DEXPER and MOOClets to provide individualized recommendations of problems for students in a MOOC, based on a student's performance on prior problems. The *Learner Interface* is provided by an LTI tool that allows the embedding of various problem "windows" inside an edX MOOC. The problems displayed in a window were native edX content, but the MOOClet allowed an external web service to decide which problem was displayed. This was enabled by the *Version Set* containing links to individual problems. Then, DEXPER allowed problem recommendation to be outsourced to an external web service. An organization provided an API to receive problem recommendations based on their implementation of Bayesian Knowledge Tracing, which obtained a user's variables from the *Learner Data Store*, such as performance on past problems and other course activities like video watching.

CONCLUSION

This paper presents the *MOOClet* requirements specification for designing software for experimentation, so that every randomized comparison of versions of a resource enables dynamic improvement and personalization of that resource. We created a proof-of-concept web service, DEXPER, that provides the backend for a MOOClet: Version Set, Learner Data Store, and a suite of Policies. We described three applications of MOOClets and DEXPER: to experiment on and personalize emails to MOOC learners, automatically improve learners' satisfaction with explanations in math problems, and adaptively tailor problems in a MOOC based on learners' past performance. For future work, we plan to improve the specification and tools by working with more instructors and researchers and verifying improvements in their practice and course content. Furthermore, we will release our tools as open source.

REFERENCES

1. Eytan Bakshy, Dean Eckles, and Michael S Bernstein. 2014. Designing and deploying online field experiments. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 283–292.
2. Peter Brusilovsky and Christoph Peylo. 2003. Adaptive and intelligent web-based educational systems. *International Journal of Artificial Intelligence in Education (IJAIED)* 13 (2003), 159–172.
3. Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M. Henne. 2009. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery* 18, 1 (2009), 140–181.
4. Jacob Whitehill, Joseph Jay Williams, Glenn Lopez, Cody Austun Coleman, and Justin Reich. 2015. Beyond prediction: First steps toward automatic intervention in MOOC student stopout. *Available at SSRN 2611750* (2015).
5. Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z Gajos, Walter S Lasecki, and Neil Heffernan. 2016. AXIS: Generating Explanations at Scale with Learnersourcing and Machine Learning. In *Proceedings of the Third (2016) ACM Conference on Learning at Scale*. ACM, 379–388.