Наука и Образование. МГТУ им. Н.Э. Баумана. Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 11. С. 426–442.

DOI: **10.7463/1115.0825497**

Представлена в редакцию: 02.10.2015
Исправлена: 21.10.2015

УДК 004.42, 378.14

Метод моделирования задач для организации автоматизированной проверки ответов студентов в системах управления обучением

Гурченков А. А.[1,*], Спиридонов Р. С.[2]

[*] challenge2005@mail.ru

[1]МГТУ им. Н.Э. Баумана, Москва, Россия
[2]МФТИ, Долгопрудный, Россия

Разработан метод, позволяющий создавать произвольные типы ввода ответов и интегрировать различные алгоритмы для их оценки в системы электронного обучения. Создана модель и жизненный цикл объектов типа «алгоритмическая задача». Использование предложенных решений повышает эффективность преподавательской работы, благодаря автоматизированию проверки домашних работ и тестов. Преимуществом подхода является поддержка генерации алгоритмических вариаций задач, что позволяет преподавателю выдавать уникальные версии одной задачи всем студентам группы. Выполнена практическая реализация предложенного подхода в виде программного Java модуля, реализующего протокол, который позволяет производить интеграцию с существующими системами управления обучением. Обеспечена отказоустойчивость системы путем сохранения состояния объектной модели в базе данных.

Ключевые слова: электронное обучение, системы управления обучением, алгоритмы оценки, типы вопросов, Java, REST

Введение

Развитие Интернет-технологий и клиент-серверной архитектуры Web 2.0 позволяет переосмыслить подходы к решению целого ряда проблем, с которыми сталкиваются современные организации и институты. Безусловно, это касается и сферы образования, где в последние годы активно разрабатываются и внедряются системы электронного обучения (ЭО). Крупные международные издательства учебной литературы перепрофилируются от продажи печатных изданий (учебников) к продаже прав доступа к системам управления обучением (СУО) промышленного масштаба. Университеты, в свою очередь, лицензируют права доступа к этим системам у издательств, чтобы преподавательский состав мог зарегистрироваться в системе и взаимодействовать в ней со своими группами студентов [1]. К современным СУО предъявляются следующие требования.

1. *Модульность.*
   - Обеспечение высокой доступности системы под большой нагрузкой (порядка тысяч одновременных пользователей). Есть возможность

развертывания каждого модуля на отдельном сервере кластера и балансирования его нагрузки.

- Повышение эффективности процесса разработки приложения: каждая команда программистов работает над отдельным модулем с четко определенным интерфейсом взаимодействия с другими модулями в рамках концепции сервисно-ориентированной архитектуры (SOA).

2. *Кросс-платформенность.* Организация распределенного доступа к СУО с любых устройств, в т.ч. планшетов и мобильных устройств.
3. *Автоматизация процессов.* Повышение эффективности учебного процесса путем автоматизации рутинных процессов преподавательской деятельности.
4. *Быстрый поиск.* Создание интегрированной базы знаний и организация быстрого поиска обучающих материалов.
5. *Анализ больших данных (Big Data).* Хранение больших массивов разнородных данных о действиях пользователей в СУО для дальнейшего их анализа и организации процесса адаптивного обучения.

Неотъемлемой частью как индивидуального процесса обучения, так и традиционного обучения в группе с преподавателем является решение практических задач. В российских вузах в последние годы происходило активное внедрение множества СУО, наиболее популярными из которых являются открытые системы Moodle, Ilias, Sakai и др. [2]. Эти системы предоставляют возможности создания и публикации курсов (учебных материалов), сопровождаемых тестами и задачами для самопроверки. Задачи в этих системах реализуют один из нескольких простых типов вопросов [3,4], таких как выбор из нескольких вариантов ответа, текстовый ввод, сопоставление, или ввод числа. Системы массового он-лайн обучения, такие как Coursera и Edx, также предоставляют встроенные средства автоматизированной проверки ответов, но только для простых типов ввода.

Сведение большинства сложных задач к выбору из нескольких предопределенных вариантов ответа не способствуют развитию творческих подходов к решению задач у ученика, а также не является достаточным для проверки навыков решения реальных задач [5]. Основное преимущество такого вида вопросов кроется в возможности автоматизированной проверки простых типов ввода в СУО. Таким образом, становится актуальной проблема поддержки сложных типов ввода и их автоматизированной оценки.

Итого, в существующих системах электронного обучения, в том числе активно внедряющихся в российских учебных заведениях, есть ряд недостатков:

- отсутствие сложных типов ввода, в т.ч. графиков функций, сложных математических выражений и векторных диаграмм;
- отсутствие возможности создания алгоритмических задач и тестов;
- отсутствие выделенного сервиса для поддержки отображения и проверки задач, что не дает возможность использовать единые решения и единый формат задач в различных СУО.

Стоит отметить, что при перечисленных недостатках активно-используемых СУО, в научной литературе последних лет приведено множество исследований, касающихся области автоматизированной оценки ответов студентов. Так, созданы различные методы оценки математических выражений [6,7], графиков функций [7,8], геометрических фигур [9], эссе [10] и других сложных типов ввода [11]. Однако все эти решения заточены под конкретный формат задач и носят частный характер, что мешает их повсеместному использованию в реальных системах. Отсутствует единое решение, которое позволит инкапсулировать работу с задачами, включая применение алгоритмов автоматической оценки ответов студентов, в системах электронного обучения.

1. Постановка задачи

Целью работы является формализация объектов типа «алгоритмическая задача» и реализация методов взаимодействия с ними в виде независимого программного модуля. Интеграция модуля с существующими СУО позволит расширить их функционал для поддержки сложных типов вопросов и их автоматической оценки. Архитектура модуля должна удовлетворять всем вышеизложенным требованиям, актуальным для сред ЭО.

Модуль должен предоставлять следующую функциональность:

- обращение к нему посредством определенного протокола из любой платформы, сервиса, или системы управления обучением;
- генерация веб-сервером страниц в формате, поддерживающем работу на мобильных устройствах и планшетах;
- экономия времени преподавателя посредством автоматизации выставления оценки за выполнение задания (не нужно вручную проверять работу студента), генерации различных вариаций одной и той же задачи для назначения одной группе студентов (нет необходимости подбирать уникальный вариант задачи для каждого студента в группе);
- хранение задач и тестов в унифицированном формате XML с мета-данными для каждой задачи;
- сохранение данных о состоянии текущей задачи и действиях студентов в базе данных для обеспечения отказоустойчивости и последующего анализа данных.

2. Теоретический подход к решению задачи

В разделе 2.1 вводится понятие «алгоритмическая задача» (АЗ) как ключевого объекта моделирования. Каждая АЗ имеет множество состояний, которые она последовательно принимает в ходе своего жизненного цикла. На множестве АЗ определяется функции, переводящие АЗ из одного состояния в другое. В разделе 2.2 приводится формализованная XML модель объекта АЗ. Раздел 2.3 посвящен подходу к поддержке различных вариантов одной задачи путем синтеза ее алгоритмов.

2.1. Модель алгоритмической задачи

Основной целью работы является создание и реализация метода для поддержки алгоритмических задач в существующих СУО. Вначале, формализуем понятие «алгоритмическая задача» (АЗ). Рассмотрим следующую модель (описание) объектов этого типа. Будем называть *алгоритмическими задачами* все элементы, принадлежащие следующему множеству *АЗ* и замкнутые относительно функций своего жизненного цикла:

$$(\mathbf{id}\ ,\ \mathbf{contents}\ \{\ text\ \&\ \{\ alg,\ tags,\ fields[N]\ \}\ \},\ \mathbf{corrects}[N]\ \&\ \{alg\},\ \mathbf{algorithm}\ ,\ \mathbf{state}[seed,\ grade]). \quad (1)$$

В этой записи приняты следующие обозначения:

- ***id*** – число, используемое для идентификации элемента множества. Для этого используется значение хэш-функции, вычисленной на конкатенации содержимого (contents), алгоритма задачи (algorithm), и правильных ответов (corrects[N]).
- ***contents { }*** – содержимое задачи, готовое к отображению конечному пользователю (например, в формате HTML5 с набором связанных JS и CSS файлов). Запись в фигурных скобках означает, что помимо текста задачи с тегами разметки (*text*), могут использоваться дополнительные элементы:
    - *alg* – переменные из алгоритма в виде тегов <alg>*имя_переменной*</alg>; значения переменных будут подставлены на этапе компиляции задачи;
    - *tags* – набор заранее определенных дополнительных элементов разметки со своим внутренним представлением (например, <graph> с форматом для объектов типа «график»); эти элементы будут в процессе обработки заменены на стандартные элементы разметки *text* для отображения задачи конечному пользователю;
    - *fields[N]* – набор из *N* (*N>0*) полей для ввода ответа, заданных в виде <field type=”*тип_ввода*”/>; каждое поле имеет свое внутреннее представление, которое задает конфигурацию одного из сложных типов ввода (например, «ввод графика»).
- ***algorithm*** – блок определения алгоритма задачи, в котором присутствуют определения переменных и операции с ними. В результате выполнения алгоритма, все переменные получают некоторые значения, которые могут быть подставлены в качестве входных данных в текст задачи (*text & {alg}*), а так же в качестве правильных ответов (*corrects[N] & {alg}*).
- ***corrects****[N]* – для каждого из *N* полей для ввода указывается один или несколько правильных ответов и метод автоматической оценки (<correct method=”*метод_оценки*”>*правильный_ответ*</correct>). Метод определяет алгоритм оценки правильности ответа студента при его сопоставлении с правильным ответом, т.е. отвечает за правила заполнения объекта state[grade].

- ***state*** – данные о состоянии задачи, представленные двумя элементами:
    - *seed* – число, характеризующее состояние скомпилированной задачи (-1 – если задача еще не была скомпилирована);
    - *grade* – данные об ответах студента, оценка их правильности (score) и обратная связь в форме подсказки, сформированной по ответу студента (rejoinder).

Определим подмножества исходного множества алгоритмических задач АЗ, которые назовем *состояниями жизненного цикла задачи*.

Назовем АЗ *созданной*, если ее ID уже вычислен, подстановки значений переменных alg не были совершены, значение состояния равно «-1» ($seed = -1$) и ответ студента еще не поступил ($grade = \varnothing$):

$$(\mathbf{id}, contents\{text \;\&\; \{alg, tags, fields[N]\}\}, corrects[N] \;\&\; \{alg\}, algorithm, \mathbf{-1}). \tag{2}$$

Назовем созданную АЗ *скомпилированной*, если в ней не присутствует переменных (элементов alg), а вместо ссылок на переменные подставлены их значения. Скомпилированную АЗ отличает, таким образом, отсутствие возможности использования тегов переменных <alg> и заданное значение seed, отличное от -1:

$$(id, contents\{text \;\&\; \{tags, fields[N]\}\}, corrects[N], algorithm, state[\mathbf{seed}]). \tag{3}$$

Назовем скомпилированную АЗ *готовой к отображению*, если в ее содержимом (contents) не присутствует элементов alg, $tags$, $fields[N]$:

$$(id, contents\{text\}, corrects[N], algorithm, state[seed]). \tag{4}$$

Наконец, назовем скомпилированную АЗ *пройденной*, если на нее уже поступил ответ студента, и ему была присвоена оценка ($grade \neq \varnothing$):

$$(id, contents\{text \;\&\; \{tags, fields[N]\}\}, corrects[N], algorithm, state[seed, \mathbf{grade}]). \tag{5}$$

Описанная структура данных задачи из $АЗ_{created}$ может формироваться на основе файла данных XML, синтаксис которого более детально рассмотрен в следующем разделе 2.2.

Функции жизненного цикла алгоритмической задачи имеют следующий вид.

- **create():** XML $\rightarrow АЗ_{created}$ – создание задачи, т.е. создание хранимого в памяти объекта из тестовых данных определенного формата. Этот шаг включает следующие действия:
    - анализ текстового содержимого для заполнения структуры объекта «алгоритмическая задача» ($АЗ_0$);

- вызов хэш-функции для получения уникального идентификатора (ID).

- **compile()**: $АЗ_{created} \to АЗ_{compiled}$ – компиляция задачи, т.е. вычисление всех значений переменных алгоритма и их подстановка в текст задачи и правильные ответы. Этот шаг включает следующие действия:
  - генерация псевдослучайных чисел для базисных переменных алгоритма: значение состояния (*seed*) принимает значение зерна алгоритма генерации псевдослучайных чисел [12];
  - компиляция алгоритма;
  - подстановка вычисленных значений всех переменных вместо *<alg>имя_переменной</alg>* в тексте (text) и в тексте правильных ответов (corrects[N]).
- **render():** $АЗ_{created} \to АЗ_{rendered}$ – отображение задачи для конечного пользователя путем вывода ее содержимого *text* (по определению, *text* содержит данные, готовые к отображению пользователю – например, данные в формате HTML для отображения на любой платформе, поддерживающей браузер). Этот шаг включает следующие действия:
  - обработка данных конфигураций элементов контента (tags) и полей для ввода (fields[N]) для получения выходного формата в формате, готовом для отображения пользователю (text);
  - для каждого поля fields[N], выходной формат данных для отображения должен включать в себя интерфейс для отправки ответа на сервер (в виде закодированной строчки).
- **submit():** $АЗ_{rendered} \to АЗ_{submitted}$ – оценка введенного студентом ответа, генерация обратной связи (подсказки), сохранение этой информации в объекте «задача». Этот шаг включает следующие действия:
  - прием и обработка закодированной строки ответа от клиента (студента) для каждого поля field[N];
  - сравнение строки ответа студента со строкой правильного ответа при помощи указанного в corrects[N] алгоритма оценки – генерация оценки (score) и обратной связи в виде подсказки (rejoinder) студенту о том, что в его или ее ответе было ошибочным;
  - заполнение state[grade] данными с предыдущего шага.

2.2. Структура алгоритмической задачи: общий формат XML

Для представления элементов множества алгоритмических задач ($АЗ$) в текстовой форме разработан формат XML, который может использоваться для хранения задачи на диске или в базе данных. Отличительными свойствами формата является свойства *расширяемости* и *портируемости*. Каждый тег верхнего уровня XML схемы представляет из себя основные блоки структуры объекта «алгоритмическая задача»: содержимое (contents); алгоритм (algorithm); правильные ответы (corrects[N]). ID и state в XML не сохраняются, т.к. они вычисляются в процессе жизненного цикла, т.е. в процессе работы приложения.

Рассмотрим формат XML задачи на примере (рис. 1).

<item> – корневой тег (единый корневой тег необходим в соответствии со спецификацией формата XML).

<value name='algorithm'> – блок определения алгоритма для вариации задачи; в этом блоке задаются входные данные как переменные величины, и на их основе вычисляется ответ, ожидаемый от студента. Таким образом, содержимое этого блока фактически описывает алгоритм решения задачи как зависимость вектора результатов от вектора входных данных: $\overrightarrow{\text{ответ}} = f(\overrightarrow{\text{входные данные}})$ (хотя допустим и метод кодирования задачи «от обратного», когда по ответу находятся подходящие входные данные).

<value name='specification'> – блок содержимого текста вопроса, представляющий собой HTML5 код со вставками специальных зарезервированных тегов, обозначающих различные типы контента. Примерами таких зарезервированных тегов являются <field> (тип вопроса) и <alg> (вставка значения переменной).

<value name='correctN'> – блок с описанием правильного ответа к полю <field> соответствующего порядкового номера *N*. Здесь автор задачи описывает правильный (ожидаемый) ответ и метод оценки, т.е. алгоритм сравнения ответа студента с правильным ответом для определения их эквивалентности.

Расширяемость обеспечивают следующие возможности приложения.

- Добавление новых типов вопросов в (fields[N]):
  <field **type="{тип_вопроса}"**>.
- Добавление новых алгоритмов оценки в (corrects[N]):
  <correct **method="{алгоритм _оценки}"**>…</correct>.
- Добавление новых типов контента в (contents[tags]):
  <value name="specification ">…**<{тип_контента}/>**…</value>.
- Создание новых функций в алгоритмах (algorithm):
  <value name="algorithm">… var b = **new_function(a)**; …</value >.

```
<item>
 <value name='algorithm'>
 <!-- ALGORITHM DEFINITION -->
 var a=real(1,5,0.1); <!-- ALGORITHMIC VARIABLE -->
 var b=integer(1,5,1);
 var c=a+b;
 </value>
 <value name='specification'>
 <!-- CUSTOM HTML WITH ALGORITHMIC VARIABLES,
 QUESTION AND FIELDS -->
 How much is
 <alg>a</alg>+<alg>b</alg>?
 <field type='number'/> <!-- FIELD 1 -->
 </value>
 <value name='correct1'>
 <!-- CORRECT ANSWER FOR FIELD 1 -->
 <correct method='string'>
 <alg>c</alg>
 </correct>
 </value>
</item>
```

Рис 1. Формат XML алгоритмической задачи.

2.3. Синтез алгоритмов задачи

Алгоритм задачи задается в блоке <value name=”algorithm”>. Результат выполнения алгоритма – это сгенерированные значения объявленных в нем переменных, которые используются в задаче как (1) входные данные для условия и (2) результаты (правильные ответы). Вставка значений переменных осуществляется с помощью тега <alg> следующим образом:

<alg>{имя_переменной}**</alg>**.

Переменная в алгоритме объявляется с использованием ключевого слова var, оператора присваивания и выражения в правой части (выражение может использовать числа, строки, и переменные, значения которых вычислены до текущей строки алгоритма):

var {имя_переменной} = {выражение}.

Синтез алгоритма производится с помощью двух типов переменных.

Базисные переменные – переменные, значения которых генерируются псевдослучайным образом на основе «зерна» (seed) с помощью следующих функций:

- oneof(x1, x2, …) – значение из фиксированного набора значений (могут быть числа или строки);
- real(min, max, step) [или integer(min, max, step)] – вещественное [или целое] значение в диапазоне [min, max], кратное шагу step.

Вычисляемые переменные – переменные, которые вычисляются на основе базисных (зависят от них).

Уникальность вариации обеспечивается использованием алгоритма генерации псевдослучайных чисел линейно-конгруэнтным методом [12]. Использование именно псевдослучайных чисел обуславливается необходимостью сохранять состояние задачи. Это свойство может использоваться, например, для обеспечения отказоустойчивости: в случае «падения» сервера во время пользовательской сессии, зная зерно (seed), система сможет восстановить состояние алгоритма задачи до «падения» сервера.

Вычисление числовых выражений производится методом сортировочной станции Э. Дейкстры [13].

3. Реализация приложения

Рассмотрим реализацию приложения, которое обеспечивает жизненный цикл объектов типа «алгоритмическая задача». В разделе 3.1 рассматривается архитектура приложения и входящие в него сервисы. Раздел 3.2 описывает роли пользователей и сценарии их взаимодействия с системой. Раздел 3.3 посвящен детальному рассмотрению интерфейса программного взаимодействия, с помощью которого осуществляется работа модуля в автономном режиме и в режиме интеграции с СУО.

3.1. Высокоуровневая архитектура приложения

Рассмотрим высокоуровневое устройство приложения. Приложение состоит из трех сервисов: *сервис рендеринга задач* (Item Service), *сервис хранения контента* (Content Service) и *слой хранения состояния задач* (Persistence Layer).

Сервис рендеринга задач получает на вход XML задачи и при условии, что она описана в правильном формате, возвращает по запросу готовый HTML код для отображения пользователю. Далее, пользователь вводит и отправляет ответ на сервер, где происходит его обработка и автоматическая оценка правильности. В конечном итоге, сервис возвращает и отображает в задаче правильный ответ и подсказку, заданную автором задачи. Запрос к сервису может содержать как сам код XML задачи для отображения, так и URL из *сервиса хранения контента*.

Слой хранения состояния задач необходим, чтобы обеспечить отказоустойчивость системы путем восстановления состояния в случае «падения» сервера, а так же для хранения результатов ответов студентов и их дальнейшего анализа. В качестве этого слоя используется база данных MongoDB. Выбор этой СУБД связан с ее высокой производительностью в случае хранения ненормализованных данных для веб-приложений в формате JSON объектов [14].

Серверная часть (back-end) приложения реализована на объектно-ориентированном языке Java. На рис. 2 показана диаграмма взаимодействия классов приложения в процессе работы с REST API [15].

Реализованы следующие классы.

ItemInstance – хранит скомпилированную задачу (CompiledItem) и информацию по истории взаимодействия с этой задачей. Фактически, класс инкапсулирует состояние задачи и предоставляет функции сохранения состояния задачи в MongoDB.

CompiledItem – класс, разбирающий XML по структурам данных, которые удобно использовать в программном коде (скомпилированная задача).

ItemCompiler – класс, предоставляющий возможности компиляции задачи и алгоритма (вызов функции compile() , возвращающей CompiledItem).

ItemResolver – класс, отвечающий за получение itemXML задачи. Класс предоставляет операции с исходным кодом задачи.

Algorithm Language – набор классов лексического анализатора и компилятора алгоритма, использующегося для генерации различных вариантов одной и той же задачи для выдачи группе студентов. В XML задачи соответствует тегу *<value name=”algorithm”>*.

Question Types – набор классов для типов вопросов (по одному классу на каждый тип вопроса). В XML задачи каждый класс соответствует тегу *<field type=”{question_type}”>*.

Grading Methods – набор классов с алгоритмами автоматической оценки строк ответа, приходящих с сервера (по одному классу на каждый метод оценки). В XML задачи каждому классу соответствует тег *<value name=”correct**N**”> <correct method=”{grading_method}”/></value>*, где ***N*** – порядковый номер поля ответа в содержании задачи.

Content processing – набор классов-обработчиков произвольных тегов, отличных от стандартного набора тегов HTML, например, здесь можно определить тег <graph> и назначить на него свой обработчик, который будет транслировать содержимое этого тега сторонней библиотеке, занимающейся рисованием графиков.

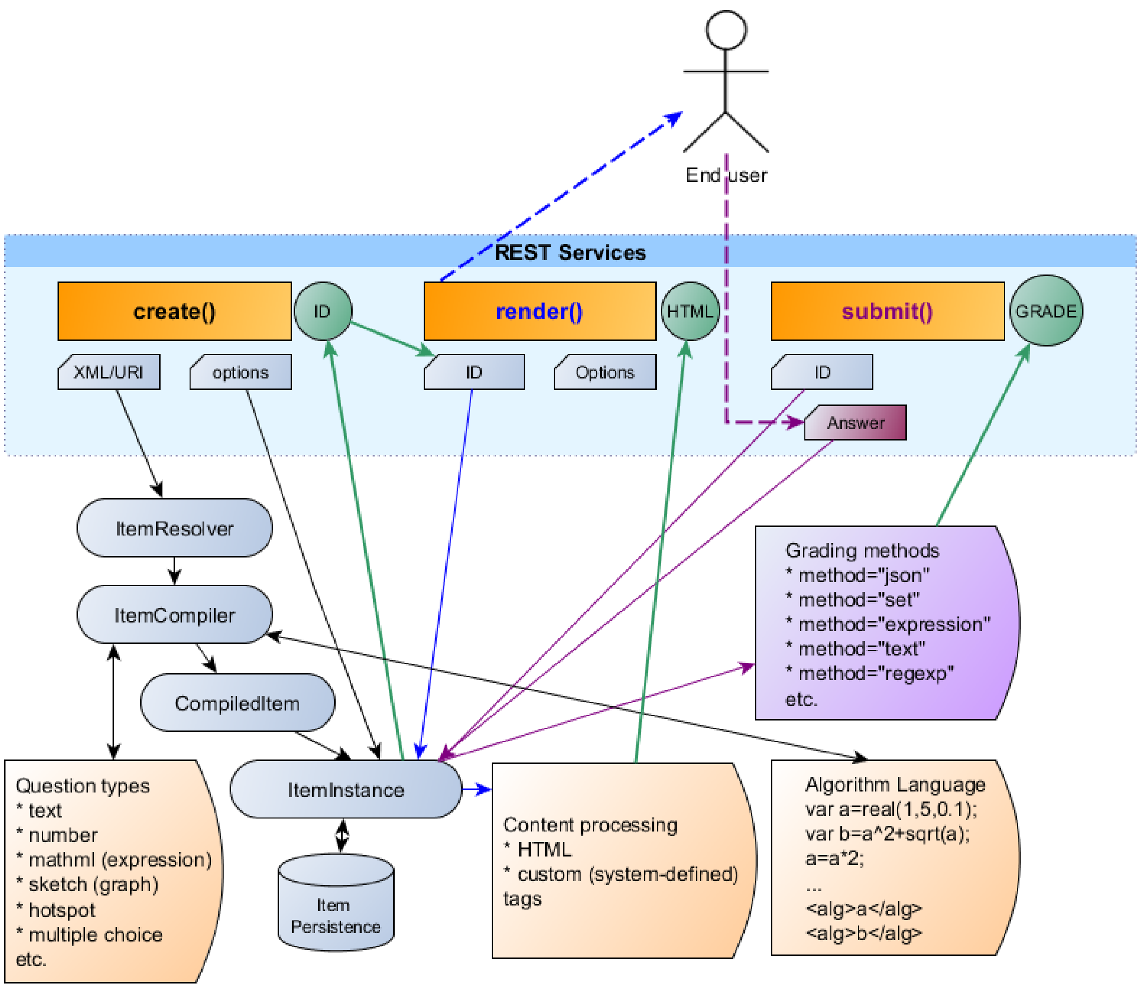

Рис. 2. Устройство модуля для отображения и оценки задач. Интерфейс программного взаимодействия (API).

3.2. Порядок работы пользователя с приложением

Использование модуля предполагает наличие трех видов пользователей предоставляемой функциональности.

Автор контента: создает XML код задачи и необходимые мультимедиа ресурсы, загружает их в сервис хранения контента.

СУО (система управления обучением, в которую интегрирован модуль): в нужном порядке вызывает функции API, т.е. функции создания и компиляции задачи (create), отображения задачи (render), и получения результатов оценки ответа студента (answer).

Студент: взаимодействует с системой через браузер после отображения задачи путем отправления на сервер своего ответа.

Для обеспечения возможности автономного тестирования модуля, необходимо компенсировать отсутствие системы для интеграции (СУО). Для этого разработан набор тестовых страниц на базе технологии JSP (Java Server Pages) [16], которые обеспечивают вызов необходимых функций жизненного цикла и предоставляют пользователю простой интерфейс для вставки XML задачи, ее загрузки, отображения и получения результатов оценки правильности ответа.

Выбор технологии JSP обусловлен тем, что она позволяет быстро создавать динамические страницы на основе существующей бизнес-логики приложения. JSP представляет собой реализацию шаблонов страниц HTML со вставками Java-кода (в специальных тегах <% %>), объявления (<%! %>) и вставки значений (<%= %>) переменных, которые на этапе обработки HTTP запроса преобразуется в код сервлета, обрабатывающий этот же запрос.

3.3. Описание интерфейса программного взаимодействия (API)

Взаимодействие с разработанным приложением происходит путем вызова API функций, совпадающим с определенными в разделе 2.1 функциями жизненного цикла объекта «алгоритмическая задача». Единственным отличием является реализация функции *String[] create(String[] itemXML)*, которая совмещает в себе сразу две функции создания create() и компиляции compile() задачи, а получаемый в результате ее вызова объект хранит как исходную $АЗ_{created}$, так и скомпилированную $АЗ_{compiled}$ с подставленными значениями алгоритмических переменных.

Вызов функций происходит по протоколу REST [15], что позволяет обращаться к сервису как развернутом на отдельном веб-сервере, так и из приложения.

API вызов по протоколу REST есть ни что иное, как обращение к соответствующему обработчику запроса по URL с передачей данных по протоколу HTTP. В отличие от других более сложных API протоколов, таких как SOAP, CORBA, или RPC, протокол REST не содержит отдельного формата для передачи данных и позволяет обращаться к функциям как к страницам веб-сайта. В параметрах HTTP запроса указываются параметры API функций вместе с URL: …index.jsp**?id=617yehj4l7810**. В качестве обработчика запросов выступают соответствующие JSP страницы (например, create.jsp для запроса create). JSP страницы на этапе обработки запроса транслируются в сервлеты (обработчики интернет-запросов) при помощи контейнера сервлетов Tomcat [17].

Возвращаемые значения функций render() и submit() выполнены в формате JSON [18], который широко используется для передачи множества данных между клиентом (браузером) и сервером.

Рассмотрим параметры и возвращаемые значения функций API.

String[] create(String[] itemXML, [options]) – подготовка задачи, ее компиляция и получение идентификатора (ID). *Возвращаемое значение* есть уникальный идентификатор задачи, генерируемый функцией получения полиномиального хэш-кода строки itemXML.hashCode() [19].

JSON render(String[] id, [options]) – вернуть HTML код раннее подготовленной задачи с идентификатором ID. *Возвращаемое значение* в формате JSON содержит HTML код страницы для отображения пользователю, а так же дополнительные поля со списком используемых ресурсов (JS, CSS). Пример возвращаемого значения:

{ id: "(itemID)", html: "(rendered item)", js: ["first.js", "second.js"], css: ["first.css"] }.

JSON answer(String answer, String verb) – отправление ответа студента на сервер, получение обратной связи (оценки, подсказки). В качестве входного параметра функции используется строка answer, в которой содержится перечисление ответов студента в каждое из полей задачи через разделитель "&". Например, строка answer ="1=green&2=5" означает, что в поле «1» был введен ответ "green", а в поле «2» – число 5. *Возвращаемое значение* в формате JSON содержит нормированную оценку (score) правильности ответа студента от 0 до 1 (0 – ответ неверный, 1 – ответ верный, промежуточные значения – ответ частично верный). Это значение так же содержит подсказку или объяснение (rejoinder), т.е. обратную связь от системы, почему ответ был дан неверно. Соответствующая подсказка пишется автором задачи в XML. Пример возвращаемого значения:

{ score: 0.5, rejoinder: ["Your answer was partially correct."],
subGrades: { "1": { score: 1.0, rejoinder: ["Correct."] }
"2": { score: 0.0, rejoinder: ["Incorrect."] } } }.

4. Заключение

Предложен метод моделирования объектов типа «алгоритмическая задача», который позволяет значительно повысить эффективность преподавательской работы благодаря автоматизированной проверке задач в домашних работах и тестах. Метод позволяет использовать существующие разработки в области автоматической оценки ответов и интегрировать эти решения в рамках единого сервиса для интеграции с существующими СУО.

В отличие от множества существующих решений, используемых для автоматизации проверки ответов, предложен целостный подход, который позволяет интегрировать эти и другие решения в существующие системы электронного обучения. Эта возможность достигается благодаря выделению функциональности по работе с задачами в отдельный программный модуль, работающий с унифицированным форматом XML для задач в системах электронного обучения. Дополнительным преимуществом разработанной модели является поддержка генерации различных вариантов одной и той же задачи путем синтеза ее алгоритма. Данный подход позволяет автоматизировать процесс формирования уникальных вариантов домашних заданий и контрольных работ для групп студентов.

Выполнена практическая реализация метода в виде независимого Java модуля, который может быть интегрирован с существующими системами электронного обучения посредством REST API протокола. В случае отсутствия системы для интеграции, реализован набор тестовых страниц на технологии JSP для работы в автономном режиме. Разработанная архитектура Java приложения для отображения и оценки выполнения задач студентами имеет ряд преимуществ:

- поддерживается алгоритмический язык для генерации различных вариантов одной и той же задачи;
- разработанный формат XML алгоритмической задачи позволяет унифицировать отображение задачи на различных платформах;
- реализованный REST протокол позволяет интегрировать приложение с различными системами управления обучением;
- интеграция с СУБД MongoDB обеспечивает отказоустойчивость системы и возможность проведения аналитики;
- реализованная иерархия классов заточена под добавление новых типов ввода ответов студентом и алгоритмов их оценки, позволяя исследователю не вникать в детали реализации системы.

Выделим следующие направления дальнейших исследований на основе разработанной архитектуры.

1) Создание сложных типов ввода ответов, таких как: графики, математические выражения, векторные диаграммы, химические молекулы и др.
2) Создание алгоритмов оценки таких ответов.
3) Расширение обратной связи студенту генерацией подсказок, указывающих, где именно в ответе была допущена ошибка, но не выдающие правильный ответ.
4) Поддержка развертывания модуля в кластере из нескольких серверов. Оптимальная сеть может быть построена с помощью минимаксного критерия [20,21].

Список литературы

1. E-Learning Market Trends & Forecast 2014 – 2016 Report. A report by Docebo. Docebo, 2014. 48 p.
2. Кривичев А.И., Сидоренко В.Н. Использование открытых систем управления обучением в вузах // Всероссийская научно-практическая конференция «Развивающие информационные технологии в образовании: использование учебных материалов нового поколения в образовательном процессе» (ИТО-Томск-2010): матер. Томск, 2010. С. 270-273. Режим доступа: http://ito.edu.ru/2010/Tomsk/III/III-0-14.html (дата обращения 01.10.2014).
3. Question types // Moodle 2.9 documentation: website. Режим доступа: https://docs.moodle.org/29/en/Question_types (дата обращения 23.05.2015).

4. SAMigo Features // Sakai Wiki: website. Режим доступа: https://confluence.sakaiproject.org/display/SAM/SAMigo+Features (дата обращения 23.05.2015).
5. Livne N.L., Livne O.E., Wight C.A. Automated Error Analysis through Parsing Mathematical Expressions in Adaptive Online Learning // Proc. of the E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education. Chesapeake, VA: Publ. of Association for the Advancement of Computing in Education (AACE), 2006. P. 1321-1325.
6. Fife J.H. Automated scoring of mathematics tasks in the common core era: enhancements to m-rater in support of CBAL™ mathematics and the common core assessments // ETS Research Report Series. 2013. Vol. 2013, iss. 2. P. 1-35. DOI: 10.1002/j.2333-8504.2013.tb02333.x
7. Bennett R.E., Morley M., Quardt D. Three Response Types for Broadening the Conception of Mathematical Problem Solving in Computerized Tests // Applied Psychological Measurement. 2000. Vol. 24, no. 4. P. 294-309. DOI: 10.1177/01466210022031769
8. Masters J. Automated Scoring of an Interactive Geometry Item: A Proof-of-Concept // The Journal of Technology, Learning and Assessment. 2010. Vol. 8, no. 7.
9. Shermis M.D., Burstein J. Handbook of automated essay evaluation: Current applications and new directions. Routledge, 2013.
10. Williamson D.M., Bejar I.I., Mislevy R.J. Automated scoring of complex tasks in computer-based testing. Manwah, New Jersey: Laurence Erlbaum Associates, Inc., 2006.
11. Бараш Л. Алгоритм AKS проверки чисел на простоту и поиск констант генераторов псевдослучайных чисел // Безопасность информационных технологий. 2005. № 2. С. 27-38.
12. Norvell T. Parsing Expressions by Recursive Descent. 2001 // Faculty of Engineering and Applied Science at Memorial University: website. Режим доступа: http://www.engr.mun.ca/~theo/Misc/exp_parsing.htm (дата обращения 23.05.2015).
13. Pirtle M. MongoDB for Web Development. Addison-Wesley Professional, 2011. 360 p.
14. Richardson L., Ruby S. RESTful Web Services. O'Reilly Media, 2007. 545 p.
15. Hunter J., Crawford W. Java Servlet Programming. 2nd ed. O'Reilly Media, 2001. 782 p.
16. Перри Б.У. Развертывание одиночной страницы JSP под Tomcat // Java сервлеты и JSP. Сборник рецептов: пер. с англ. 2-е изд. М.: КУДИЦ-Образ, 2006. С. 41-43.
17. JSON // Википедия: сайт. Режим доступа: https://ru.wikipedia.org/wiki/JSON (дата обращения 28.05.2015).
18. Katz J., Lindel Y. Collision-Resistant Hash Functions // In book: Introduction to Modern Cryptography. CRC Press, 2014. P. 121-131.
19. Mironov A.A., Tsurkov V.I. Open transportation models with a minimax criterion // Doklady Mathematics. 2001. Vol. 64, no. 3. P. 374-377.
20. Мокряков А.В., Цурков В.И. Восстановление 2-комплексов по целочисленному неотрицательному вектору // Автоматика и телемеханика. 2011. № 12. С. 130-143.

Science & Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

Science and Education of the Bauman MSTU, 2015, no. 11, pp. 426–442.

DOI: **10.7463/1115.0825497**

Received: 02.10.2015
Revised: 21.10.2015

Method of Modeling Questions for Automated Grading of Students' Responses in E-Learning Systems

A.A. Gurchenkov[1,*], R.S. Spiridonov[2]

[*] challenge2005@mail.ru

[1]Bauman Moscow State Technical University, Moscow, Russia
[2]Moscow Institute of Physics and Technology State University, Dolgoprudny, Russia

Keywords: e-learning, learning management systems, automated scoring, question types, Java, REST

Introduction. Problem relevance. The capability to check a solution of practical problems automatically is an important functionality of any learning management system (LMS). Complex types of questions, implying creative approach to problem solving are of particular interest. There are a lot of studies presenting automated scoring algorithms of students' answers, such as mathematical expressions, graphs, molecules, etc. However, the most common types of problems in the open LMS that are being actively implemented in Russian and foreign universities (Moodle, Sakai, Ilias etc.) remain simple types of questions such as, for example, multiple choice.

Study subject and goal. The purpose of study is to create a method that allows integrating arbitrary algorithms of answer scoring into any existing LMS, as well as its practical implementation in the form of an independent software module, which will handle questions in LMS.

Method. The model for objects of type "algorithmic question" is considered. A unified format for storing objects of this type, allowing keeping their state, is developed. The algorithm is a set of variables, which defines the responses versus input data (or vice versa). *Basis variables* (input) are selected pseudo-randomly from a predetermined range, and based on these values *resulting variables* (responses) are calculated. This approach allows us to synthesize variations of the same question. State of the question is saved by means of "seed" of pseudo-random number generator. A set of algorithmic problems was used to build the lifecycle management functions, namely: initialization *create ()*, rendering *render ()*, and evaluation *answer ()*. These functions lay the foundation for the Application Program Interface (API) and allow us to control software module responsible for the questions in LMS.

Practical results. This study is completed with the implementation of software module responsible for mapping the interaction with the student and automated scoring of algorithmic questions. The module supports both standalone mode using the test pages, and the mode of integration with any LMS. Special attention is given to fault tolerance of the module by allowing a

continuous state preservation of algorithmic questions in the database. The developed module architecture enables the integration of any types of input and scoring algorithms in LMS. An advantage of the suggested approach is the capability to generate a number of question options, which is necessary to form the homework or tests to be given to groups of students.

Application. The results can be used to extend the functionality of e-learning systems for the inclusion of complex question types that support automated scoring of student answers.

Further studies. Further research based on this method may include the integration of new student's answer input methods and their scoring algorithms, as well as creating other algorithms for the system feedback loop implemented in *answer ()* function. For example, one of the urgent problems consists in generation of hints based on the analysis of students' wrong responses.

References

1. *E-Learning Market Trends & Forecast 2014 – 2016 Report. A report by Docebo*. Docebo, 2014. 48 p.
2. Krivichev A.I., Sidorenko V.N. Usage of open-source learning management systems in universities. *Vserossiiskaya nauchno-prakticheskaya konferentsiya "Razvivayushchie informatsionnye tekhnologii v obrazovanii: ispol'zovanie uchebnykh materialov novogo pokoleniya v obrazovatel'nom protsesse" (ITO-Tomsk-2010): mater*. [Proc. of the All-Russian scientific-practical conference "Educational Information Technologies in Education: the use of a new generation of educational materials in the educational process" (ITO-Tomsk 2010)]. Tomsk, 2010, pp. 270-273. Available at: http://ito.edu.ru/2010/Tomsk/III/III-0-14.html , accessed 01.10.2015. (in Russian).
3. *Question types*. Moodle 2.9 documentation: website. Available: https://docs.moodle.org/29/en/Question_types , accessed 23.05.2015.
4. *SAMigo Features*. Sakai Wiki: website. Available at: https://confluence.sakaiproject.org/display/SAM/SAMigo+Features , accessed 23.05.2015.
5. Livne N.L., Livne O.E., Wight C.A. Automated Error Analysis through Parsing Mathematical Expressions in Adaptive Online Learning. *Proc. of the E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*. Chesapeake, VA, Publ. of Association for the Advancement of Computing in Education (AACE), 2006. P. 1321-1325.
6. Fife J.H. Automated scoring of mathematics tasks in the common core era: enhancements to m-rater in support of CBAL™ mathematics and the common core assessments. *ETS Research Report Series*, 2013, vol. 2013, iss. 2, pp. 1-35. DOI: 10.1002/j.2333-8504.2013.tb02333.x
7. Bennett R.E., Morley M., Quardt D. Three Response Types for Broadening the Conception of Mathematical Problem Solving in Computerized Tests. *Applied Psychological Measurement*, 2000, vol. 24, no. 4, pp. 294-309. DOI: 10.1177/01466210022031769
8. Masters J. Automated Scoring of an Interactive Geometry Item: A Proof-of-Concept. *The Journal of Technology, Learning and Assessment*, 2010, vol. 8, no. 7.

9. Shermis M.D., Burstein J. *Handbook of automated essay evaluation: Current applications and new directions*. Routledge, 2013.
10. Williamson D.M., Bejar I.I., Mislevy R.J. *Automated scoring of complex tasks in computer-based testing*. Manwah, New Jersey, Laurence Erlbaum Associates, Inc., 2006.
11. Barash L. AKS algorithm of checking for prime numbers and search of constants of pseudo-random number generators. *Bezopasnost' informatsionnykh tekhnologii = Information technology security*, 2005, no. 2, pp. 27-38. (in Russian).
12. Norvell T. *Parsing Expressions by Recursive Descent*. 2001. Faculty of Engineering and Applied Science at Memorial University: website. Available at: http://www.engr.mun.ca/~theo/Misc/exp_parsing.htm , accessed 23.05.2015.
13. Pirtle M. *MongoDB for Web Development*. Addison-Wesley Professional, 2011. 360 p.
14. Richardson L., Ruby S. *RESTful Web Services*. O'Reilly Media, 2007. 545 p.
15. Hunter J., Crawford W. *Java Servlet Programming*. 2nd ed. O'Reilly Media, 2001. 782 p.
16. Perry B.W. Recipe 2.4. Deploying an Individual JSP on Tomcat. In book: *Java Servlet and JSP Cookbook*. O'Reilly, 2004, pp. 29-30. (Russ. ed.: Perry B.W. Razvertyvanie odinochnoi stranitsy JSP pod Tomcat. In book: *Java servlety i JSP. Sbornik retseptov*. Moscow, KUDITs-Obraz Publ., 2006, pp. 41-43.).
17. *JSON*. Wikipedia, the free encyclopedia: website. Available at: https://en.wikipedia.org/wiki/JSON , accessed 28.05.2015.
18. Katz J., Lindel Y. Collision-Resistant Hash Functions. In book: *Introduction to Modern Cryptography*. CRC Press, 2014, pp. 121-131.
19. Mironov A.A., Tsurkov V.I. Open transportation models with a minimax criterion. *Doklady Mathematics*, 2001, vol. 64, no. 3, pp. 374-377.
20. Mokryakov A.V., Tsurkov V.I. Reconstructing 2-complexes by a nonnegative integer-valued vector. *Avtomatika i telemekhanika*, 2011, no. 12, pp. 130-143. (English version of journal: *Automation and Remote Control*, 2011, vol. 72, no. 12, pp. 2541-2552. DOI: 10.1134/S0005117911120095).