УДК 004.42, 378.14

Р.С. Спиридонов

АРХИТЕКТУРА ПРИЛОЖЕНИЯ ДЛЯ ПОДДЕРЖКИ РАЗЛИЧНЫХ ТИПОВ АЛГОРИТМИЧЕСКИХ ЗАДАЧ И ИХ АВТОМАТИЗИРОВАННОЙ ПРОВЕРКИ В СИСТЕМАХ ЭЛЕКТРОННОГО ОБУЧЕНИЯ

Существующие открытые системы управления обучением (СУО) не имеют встроенных возможностей, достаточных для поддержки продвинутых типов ввода и их автоматизированной оценки. Современные исследования предлагают множество подходов и алгоритмов автоматической оценки ответов для различных типов ввода. Однако эти решения заточены под конкретный формат задачи, не всегда совпадающий с форматом конкретной системы, что мешает их интеграции с реальными СУО. Целью данного исследования является создание метода, который позволил бы интегрировать сложные типы ввода ответов и алгоритмы их оценки в различные СУО. Для решения этой задачи разработана модель объектов типа «алгоритмическая задача». Жизненный цикл каждой задачи управляется функциями создания (create), компиляции (compile), отображения (render), и отправки (submit) задачи для оценки на сервер. Предложен формат для хранения состояния модели, что позволяет унифицировать решение для различных устройств и систем. Выполнена практическая реализация предложенного подхода в виде программного Java модуля, реализующего интерфейс программного взаимодействия (API), который позволяет производить интеграцию с существующими СУО. Отказоустойчивость модуля обеспечена путем сохранения состояния объектной модели в базе данных. Преимуществом предложенного подхода является поддержка алгоритмической вариации задачи, что позволяет преподавателю выдавать уникальные варианты одной задачи всем студентам группы.

Электронное обучение; Системы управления обучением; алгоритмы оценки; типы вопросов; Java; REST.

R.S. Spiridonov

APPLICATION ARCHITECTURE FOR SUPPORT OF MULTIPLE QUESTION TYPES AND THEIR AUTOMATED GRADING IN E-LEARNING SYSTEMS

Existing open-source Learning Management Systems (LMS) do not have sufficient functionality embedded to allow creation of complex question types and their automated grading. At the same time, a number of studies has been carried out that suggest a lot of methods for automated scoring of complex question types in different disciplines. However, the link is missing between the two that will make it possible to integrate the new scoring methods into the LMS. This study presents extensible solution that allows integrating different input types and adding algorithms for their automated grading. A model and lifecycle for objects of "algorithmic question" type are developed. The lifecycle of each question is managed by functions create(), compile(), render(), and submit() for automated grading on server side. For saving the state of the model a common format is developed that unifies item presentation on different devices and systems. The implementation of the approach is performed in the form of Java module that supports a protocol for integration with the existing LMS. Fault tolerance of the module is achieved by storing the state of object model in database. The usage of this method saves teacher's time by automating scoring of assigned homework assignments and tests. The upside of the suggested approach is the ability to generate algorithmic question variations, which allows a teacher to assign a unique version of the same task to each student in the group.

E-Learning; e-assessment; Learning Management Systems; automated scoring; question types; Java; REST.

1. Введение. Развитие Интернет-технологий и клиент-серверной архитектуры Web 2.0 позволяет переосмыслить подходы к решению целого ряда проблем, возникающих в сфере образования. Решение задач является неотъемлемой частью процесса обучения и требует особого внимания при реализации процесса обучения в системах управления обучением (СУО).

Большинство открытых СУО, используемые в университетах, такие как Moodle, Ilias, Sakai и др. [1] предоставляют возможности публикации курсов и учебных материалов, сопровождаемых тестами и задачами для самопроверки. Задачи в этих системах реализуют один из простых типов вопросов [2, 3]: в основном, это выбор из нескольких вариантов ответа. Однако задачи, сформулированные в виде выбора из нескольких предопределенных вариантов ответа, не способствуют развитию творческих подходов к решению задач у ученика, а также не прививают навыки решения реальных задач [4]. Основной причиной широкого распространения простых типов вопросов является возможность их автоматической проверки в СУО, т.е. оценки без участия преподавателя. Таким образом, становится актуальной проблема поддержки продвинутых типов ввода и их автоматизированной оценки.

В современной научной литературе рассматриваются различные подходы и методы автоматической оценки ответов для различных типов ввода. Большая часть исследований сконцентрирована вокруг подходов к автоматизации оценки эссе [5–7]. Существует так же база исследований в оценке математических выражений [8, 9] и графиков [10], равно как и других типов ответов [11].

Несмотря на наличие множества исследований по автоматизации оценки ответов студентов, в существующих СУО использование этих научных достижений на практике не представляется возможным. Разнообразие подходов приводит к необходимости создать архитектуру комплексного и расширяемого решения, которое могло бы помочь интегрировать в себе различные алгоритмы оценки и типы ввода и позволить интегрировать их с существующими СУО.

2. Цель исследования. Целью работы является формализация объектов типа «алгоритмическая задача» в СУО и реализация методов взаимодействия с ними в виде независимого программного модуля. Предложенный подход должен:

- ♦ осуществлять интеграцию продвинутых типов ввода и алгоритмов их оценки;
- ♦ предоставлять возможность создания алгоритмических задач и тестов;
- ♦ использовать единое решение и единый формат алгоритмических задач для возможности их интеграции в различных СУО.

3. Постановка задачи. Рассматривается функциональность систем электронного обучения, отвечающая за контроль знаний в виде домашних заданий или контрольных работ. Каждое задание формируется преподавателем и состоит из одной или нескольких задач. Задачи являются алгоритмическими, т.е. одна задача содержит в себе алгоритм, позволяющий варьировать входные данные и ответы так, чтобы при назначении этой задачи группе студентов, каждый студент получил бы уникальный вариант. В свою очередь, каждая задача может содержать несколько вопросов, причем каждый вопрос сопровождается полем для ввода ответа (например, ввод текста, математического выражения или графика функции). Студент решает задачи из задания и отправляет свои результаты на сервер, где происходит обработка, автоматическая оценка задачи, и сохранение результатов в СУО для последующего анализа.

Требуется разработать модель данных для представления объектов типа «алгоритмическая задача» и архитектуру программного модуля, который позволит интегрировать произвольные типы для ввода в задачах и алгоритмы автоматической оценки ответов студентов.

Требуется произвести реализацию метода в виде программного модуля, предоставляющего следующую функциональность:

- генерация веб-страниц с условием задач в формате, поддерживающем работу на мобильных устройствах и планшетах;
- повешение эффективности работы преподавателя посредством:
  1) автоматизации выставления оценки за выполнение задания (не нужно вручную проверять работу студента);
  2) 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 едином формате XML;
- сохранение данных о состоянии текущей задачи и действиях студентов в базе данных для обеспечения отказоустойчивости и последующего анализа данных.

4. Метод. ***4.1. Модель алгоритмической задачи***. Формализуем понятие «алгоритмическая задача» (АЗ). Рассмотрим следующую модель (описание) объектов этого типа. Будем называть *алгоритмическими задачами* все элементы, принадлежащие следующему множеству $AЗ$ и замкнутые относительно функций своего жизненного цикла:

(id , contents { *text & { alg, tags, fields[N] }* ***}, corrects****[N] & {alg},* ***algorithm , state****[seed, grade])*

Элемент множества АЗ

В этой записи:

- ***id*** – число, гарантирующее уникальность элемента множества. Для этого используется значение хэш-функции, устойчивой к коллизиям , вычисленной на конкатенации содержимого (contents), алгоритма задачи (algorithm), и правильных ответов (corrects[N]);
- ***contents { }*** – содержимое задачи, которое может быть отображено конечному пользователю. Запись в фигурных скобках означает, что в тексте задачи, помимо текста задачи с тегами разметки (*text*), могут использоваться дополнительные элементы, а именно:
  - *alg*: переменные из алгоритма в виде тегов <alg>*имя_переменной*</alg> – значения переменных будут подставлены при отображении 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 продвинутых типов ввода (например, «ввод графика»).
- ***algorithm*** – алгоритм 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).

На объектах множества $A3$ определены функции, которые управляют *жизненным циклом алгоритмических задач*, который схематично можно изобразить в виде следующей цепочки состояний:

$$\langle \text{XML формат задачи} \rangle \xrightarrow{create()} A3_{created} \xrightarrow{compile()} A3_{compiled} \xrightarrow{render()} A3_{rendered} \xrightarrow{submit()} A3_{submitted}$$

Сами функции определяются следующим образом:

1. **create():** $\text{XML} \to A3_{created}$ – создание задачи, т.е. создание хранимого в памяти объекта из XML, синтаксис которого более детально рассмотрен в секции 4.2. Этот шаг включает следующие действия:
   - Анализ текстового содержимого для заполнения структуры объекта «алгоритмическая задача» ($A3$);
   - Вызов хэш-функции для получения уникального идентификатора (ID);
2. **compile()**: $A3_{created} \to A3_{compiled}$ – 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: seed принимает значение зерна для алгоритма генерации псевдослучайных чисел [12];
- Подстановка вычисленных значений всех переменных вместо <*alg*>*имя_переменной*</*alg*> в тексте (text) и правильных ответах (corrects[N]);

3. **render():** $A3_{created} \to A3_{rendered}$ – отображение задачи конечному пользователю путем возвращения ее содержимого *text* (по определению, text содержит данные, готовые к отображению пользователю – например, данные в формате HTML5 с файлами JS/CSS 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 каждого поля fields[N], выходной формат данных для отображения должен включать в себя интерфейс для отправки ответа на сервер;

4. **submit():** $A3_{rendered} \to A3_{submitted}$ – 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 произвольного алгоритма оценки, определенного в corrects[N];
- 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.

4.2. Структура алгоритмической задачи: общий формат XML задачи. Для представления элементов множества алгоритмических задач ($A3$) в текстовой форме разработан расширяемый формат XML, который может использоваться для хранения задачи на диске или в базе данных. 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 (см. Код 1):

1. <*item*> – корневой тег (единый корневой тег необходим в соответствии со спецификацией формата XML);
2. <*value name='algorithm'*> – блок определения алгоритма для вариации задачи;
3. <*value name='specification'*> – блок содержимого текста вопроса, представляющий собой HTML5 код со вставками специальных зарезервированных тегов, обозначающих различные типы содержимого (например, таковыми являются теги <field> и <alg>);
4. <*value name='correctN'*> – блок с описанием правильного ответа к полю <field> соответствующего порядкового номера N;

```
<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>
```

Код 1. Формат XML алгоритмической задачи.

Расширяемость обеспечивают следующие возможности приложения:

- Возможность добавления новых типов вопросов в (fields[N]): <field **type="{тип_вопроса}"**>;

- Возможность добавления новых алгоритмов оценки в (corrects[N]): <correct **method="{алгоритм _оценки}"**>…</correct>;
- Возможность добавления новых типов контента в (contents[tags]): <value name="specification ">…**<{тип_контента}/>**…</value>;
- Возможность создания новых функций в алгоритмах (algorithm): <value name="algorithm">… var b = **new_function(a)**; …</value >;

4.3. Синтез алгоритмов задачи. Алгоритм задачи задается в блоке <value name="algorithm">. Для синтеза алгоритмов используется LL(1) грамматика, что позволяет автоматически генерировать парсер алгоритма, используя стандартные средства такие, как JavaCC [13,14]. Результат выполнения алгоритма – это вычисленные значения объявленных в нем переменных, которые используются в задаче как (1) входные данные и (2) результаты (правильные ответы). Вставка значений переменных в задачу 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 выражений:

1. *Базисные переменные* – переменные, значения которых генерируются псевдослучайным образом на основе «зерна» (seed) [12];
2. *Вычисляемые переменные* – переменные, которые вычисляются на основе базисных (зависят от них). Вычисление числовых выражений в реализации данного подхода производится методом сортировочной станции Э. Дейкстры [15].

Использование псевдослучайных чисел позволяет сохранять состояние задачи для обеспечения отказоустойчивости: в случае падения сервера во время пользовательской сессии: зная зерно (seed), система сможет восстановить генерацию алгоритма и задачи до падения сервера.

Таким образом, алгоритм задачи фактически описывает алгоритм решения задачи как зависимость вектора результатов от вектора входных данных: $\overrightarrow{\text{ответ}} = f(\overrightarrow{\text{входные данные}})$, где $\overrightarrow{\text{входные данные}}$ описаются базисными переменными, а $\overrightarrow{\text{ответ}}$ – вычисляемыми. Однако допустим и метод кодирования задачи «от обратного», когда по ответу находятся подходящие входные данные.

5. Реализация и архитектура приложения.

Рассмотрим высокоуровневое устройство приложения. Приложение состоит из трех сервисов: *сервис обработки задач* (Item Service), *сервис хранения контента* (Content Service) и *слой хранения состояния* (Persistence Layer). Серверная часть (back-end) приложения реализована на объектно-ориентированном языке Java.

Сервис обработки задач отвечает за жизненный цикл алгоритмических задач, описанный в предыдущем разделе, и предоставляет программный интерфейс (API) для работы с ними. Запрос к сервису может содержать как сам код XML задачи для отображения, так URL из *сервиса хранения контента*. *Слой хранения состояния* необходим для отказоустойчивости приложения, т.к. он позволяет восстановить состояние задачи в случае падения сервера, а так же для хранения результатов ответов студентов и их дальнейшего анализа. В качестве этого слоя используется СУБД MongoDB [16].

Использование приложения предполагает трех видов пользователей:

1. *Автор контента*: создает XML код задачи и необходимые мультимедиа ресурсы, загружает их в сервис хранения контента.
2. *СУО* (система управления обучением, в которую интегрирован модуль): в нужном порядке вызывает функции API жизненного цикла задачи, т.е. создание и компиляция (create), отображение (render), и получение результатов оценки (answer).
3. *Студент*: взаимодействует с системой через браузер после отображения задачи путем отправления на сервер своего ответа.

Работа модуля происходит путем вызова API функций, совпадающих с функциями жизненного цикла объекта «алгоритмическая задача». Вызов функций происходит по протоколу REST [17]. API вызов по протоколу REST есть ни что иное, как обращение к соответствующему обработчику запроса по URL с передачей данных по протоколу HTTP. Обработчиками REST API запросов выступают соответствующие JSP страницы (Java Server Pages) [18], принимающие параметры запросов как параметры адресной строки (URL): …index.jsp**?id=617yehj4l7810**. На этой же технологии разработан набор тестовых страниц, который позволяет работать с модулем в автономном режиме, т.е. без интеграции с СУО. На Рис. 1 показана диаграмма взаимодействия классов приложения в процессе работы с REST API. Реализованы следующие классы:

1. *ItemInstance* – хранит скомпилированную задачу (CompiledItem) и информацию по истории взаимодействия с этой задачей. Фактически, инкапсулирует состояние задачи и предоставляет функции сохранения состояния задачи в MongoDB;
2. *CompiledItem* – класс, разбирающий XML по структурам данных, которые удобно использовать в программном коде;
3. *ItemCompiler* – класс, предоставляющий возможности компиляции задачи и алгоритма (вызов функции compile(), возвращающей CompiledItem);
4. *ItemResolver* – класс, отвечающий за получение itemXML задачи, предоставляет операции с исходным кодом задачи.
5. *Algorithm Language* – набор классов лексического анализатора и компилятора алгоритма (в XML задачи соответствует тегу <*value name=”algorithm”*>);
6. *Question Types* – набор классов для типов вопросов: по одному классу на каждый тип вопроса (в XML задачи соответствует тегу <*field type=”{question_type}”*>);
7. *Grading Methods* – набор классов с алгоритмами автоматической оценки строк ответа, приходящих с сервера: по одному классу на каждый метод оценки (в XML задачи соответствует тег <*value name=”correct**N**”*> <*correct method=”{grading_method}”/*></*value*>, где **N** – порядковый номер поля ответа в содержании задачи);
8. *Content processing* – набор классов-обработчиков произвольных тегов, отличных от стандартного набора тегов HTML5.

Приложение поддерживает возможность развертывания в кластере из нескольких серверов. Построение оптимальной сети серверов может быть осуществлено на основе минимаксного критерия [19–21].

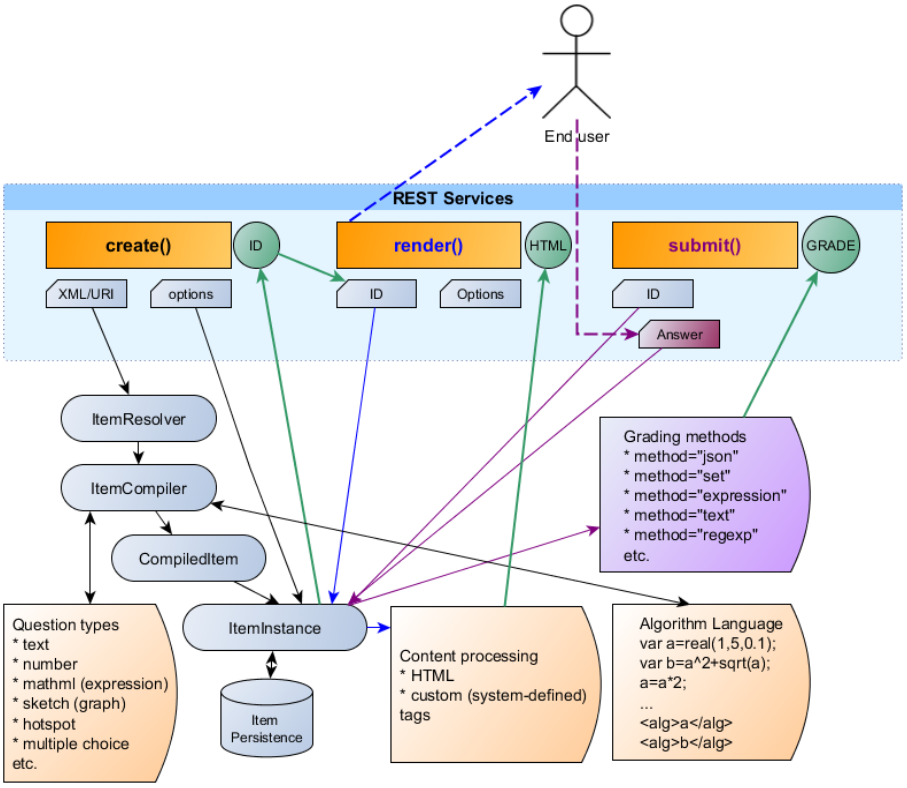

Рис. 1. Устройство модуля для отображения и оценки задач. Интерфейс программного взаимодействия (API)

Заключение. В актуальных научных исследованиях предложено множество алгоритмов автоматической оценки ответов студентов на различные сложные типы вопросов (графики, математические формулы, химические структуры и т.п.), однако все эти решения заточены под конкретный формат данных задачи, в предположениях которого работает алгоритм. Такой подход мешает осуществить повсеместную интеграцию этих алгоритмов в существующие системы управления обучением. Так, многие из существующих систем до сих пор не поддерживают сложных типов ввода [1–3].

Предложенный в этой работе метод моделирования объектов типа «алгоритмическая задача» позволяет включить произвольные типы ввода и алгоритмы оценки в единый формат задач. В рамках этой модели разработана архитектура приложения, позволяющего инкапсулировать работу с задачами и оценку ответов студентов в системах управления обучением. Выполнена реализация предложенного метода в виде Java приложения, которое может быть интегрировано с произвольной системой в виде независимого программного модуля, отвечающего за работу с задачами. Реализована поддержка автономной работы модуля на базе технологии JSP. Преимуществом разработанной модели по сравнению с существующими решениями является поддержка алгоритмической вариации задач, позволяющая преподавателю выдавать уникальную вариацию одной и той же задачи каждому студенту своей группы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Кривичев А.И., Сидоренко В.Н.* Использование открытых систем управления обучением в вузах // Информационные технологии в образовании. – 2010. – С. 270-273.
2. Question types // Moodle 2.9 documentation. – URL: https://docs.moodle.org/29/en/Question_types (дата обращения: 23.05.2015).
3. SAMigo Features // Sakai Wiki. – URL: https://confluence.sakaiproject.org/display/SAM/SAMigo+Features (дата обращения: 23.05.2015).
4. *Becker J.P., Shimada S.* The open-ended approach: A new proposal for teaching mathematics. – Reston, VA: National Council of Teachers of Mathematics, 1997.
5. *Shermis M., Burstein J.* Automated essay scoring versus human scoring: A comparative study // Journal of Technology, Learning, and Assessment. – 2007. – Vol. 6 (2).
6. *Dikli S.* An overview of automated scoring of essays // Journal of Technology, Learning, and Assessment. – 2006. – Vol. 5 (1).
7. *Shermis M.D., Burstein J.* Automated Essay Scoring: A Cross-Disciplinary Perspective. – Manwah: NJ: Lawrence Erlbaum Associates, 2003.
8. *Livne N., Livne O.E., Wight C.A.* Automated assessment of creative solutions in mathematics through comparative parsing // Creativity: A handbook for teachers. – Singapore: World Scientific publishing Co. Pte. Ltd., 2007. – P. 399-419.
9. *Livne N., Livne O.E., Wight C.A.* System and method of analyzing freeform mathematical responses // U.S. Patent №00846-25702.PROV.PCT. 2007.
10. *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. – 44 p.
11. *Masters J.* Automated Scoring of an Interactive Geometry Item: A Proof-of-Concept // Journal of Technology, Learning, and Assessment. – 2010. – Vol. 8 (7).
12. *Knuth D.E.* Section 3.2.1: The Linear Congruential Method // The Art of Computer Programming, Seminumerical Algorithms. – 1997. – Vol. 2. – P. 10-26.
13. *Knuth D. E.* On the translation of languages from left to right // Information and control. – 1965. – Vol. 8(6). – P. 607-639.
14. *Copeland T.* Generating parsers with JavaCC. – Alexandria: Centennial Books, 2007.
15. *Dijkstra E.W.* Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60 // Algol Bulletin №10. – 1961. – P. 1-20.
16. *Pirtle M.* MongoDB for Web Development. – Addison-Wesley Professional, 2011. – 360 p.
17. *Richardson L., Ruby S.* RESTful Web Services. – O'Reilly Media, 2007. – 545 p.
18. *Hunter J., Crawford W.* Java Servlet Programming. – 2 ed. – O'Reilly Media, 2001. – 782 p.
19. *Миронов А.А., Цурков В.И.* Транспортные и сетевые задачи с минимаксным критерием // Журнал вычислительной математики и математической физики. – 1995. – Т. 35, № 1.
20. *Mironov A.A., Tsurkov V.I.* Approximation and decomposition by extremal graphs // Журнал вычислительной математики и математической физики. – 1993. – Т. 33, № 2.
21. *Mironov A.A., Tsurkov V.I.* Open transportation models with a minimax criterion // Doklady Mathematics. – 2001. – Vol. 64 (3) – P. 374-377.

REFERENCES

1. *Krivichev A.I., Sidorenko V.N.* Ispol'zovanie otkrytyh sistem upravleniya obucheniem v vuzah [Usage of open-source learning management systems in universities], *Informacionnye tekhnologii v obrazovanii* [Information Technologies in Edutication]. Tomsk, 2010, pp. 270-273.
2. Question types, *Moodle 2.9 documentation*. Available at: https://docs.moodle.org/29/en/Question_types (date accessed: 23.05.2015).
3. SAMigo Features, *Sakai Wiki*. Available at: https://confluence.sakaiproject.org/display/SAM/SAMigo+Features (accessed: 23 May 2015).
4. *Becker J.P., Shimada S.* The open-ended approach: A new proposal for teaching mathematics. – Reston, VA: National Council of Teachers of Mathematics, 1997.
5. *Shermis M., Burstein J.* Automated essay scoring versus human scoring: A comparative study, *Journal of Technology, Learning, and Assessment*, 2007, Vol. 6 (2).
6. *Dikli S.* An overview of automated scoring of essays, *Journal of Technology, Learning, and Assessment*, 2006, Vol. 5 (1).

7. *Shermis M.D., Burstein J.* Automated Essay Scoring: A Cross-Disciplinary Perspective. Manwah: NJ: Lawrence Erlbaum Associates, 2003.
8. *Livne N., Livne O.E., Wight C.A.* Automated assessment of creative solutions in mathematics through comparative parsing, *Creativity: A handbook for teachers*. Singapore: World Scientific publishing Co. Pte. Ltd., 2007, pp. 399-419.
9. *Livne N., Livne O.E., Wight C.A.* System and method of analyzing freeform mathematical responses, U.S. Patent №00846-25702.PROV.PCT. 2007.
10. *Fife J.H.* Automated scoring of mathematics tasks in the common core era: enhancements to m-rater in support of cbal TM mathematics and the common core assessments, ETS Research Report Series, 2013, 44 p.
11. *Masters J.* Automated Scoring of an Interactive Geometry Item: A Proof-of-Concept, *Journal of Technology, Learning, and Assessment*, 2010, Vol. 8 (7).
12. *Knuth D.E.* Section 3.2.1: The Linear Congruential Method, The *Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 1997, pp. 10-26.
13. *Knuth D.E.* On the translation of languages from left to right, *Information and control*, 1965, Vol. 8 (6), pp. 607-639.
14. *Copeland T.* Generating parsers with JavaCC. Alexandria: Centennial Books, 2007.
15. *Dijkstra E.W.* Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60, *Algol Bulletin* No. 10, 1961, pp. 1-20.
16. *Pirtle M.* MongoDB for Web Development. Addison-Wesley Professional, 2011, 360 p.
17. *Richardson L., Ruby S.* RESTful Web Services. O'Reilly Media, 2007, 545 p.
18. *Hunter J., Crawford W.* Java Servlet Programming. 2 nd ed. O'Reilly Media, 2001, 782 p.
19. *Mironov A.A., Tsurkov V.I.* Transportnye i setevye zadachi s minimaksnym kriteriem [Transportation and network problems with minimax criterion], Zhurnal *vychislitel'noj matematiki i matematicheskoj fiziki* [Journal of computational mathematics and mathematical physics], 1995, Vol. 35, No. 1.
20. *Mironov A.A., Tsurkov V.I.* Approximation and decomposition by extremal graphs, *Zhurnal vychislitel'noj matematiki i matematicheskoj fiziki* [Journal of computational mathematics and mathematical physics], 1993, Vol. 33, No. 2.
21. *Mironov A.A., Tsurkov V.I.* Open transportation models with a minimax criterion, *Doklady Mathematics*, 2001, Vol. 64 (3), pp. 374-377.

Статью рекомендовал к опубликованию д.т.н. И.А. Матвеев.

Спиридонов Роман Сергеевич – Московский физико-технический институт; e-mail: romars@phystech.edu; 141701, г. Долгопрудный, Московская область, пер. Институтский, 9; тел.: +78634360248; аспирант.

Spiridonov Roman Sergeevich – Moscow Institute of Physics and Technology; e-mail: romars@phystech.edu; 9, Institutskiy pereulok, Dolgoprudniy, Moscow region, 141701, Russia; phone: +78634360248; postgraduate student.