Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Карельский Государственный Педагогический Университет

Студент 551 группы
А.М. Смирнов

Использование международного стандарта IMS QTI в разработке системы тестирования

Дипломная работа

Кафедра информатики
Научный руководитель:
старший преподаватель
А.С. Кюршунов

Петрозаводск
2007

Содержание

Введение

Наша образовательная система с незначительным отставанием от заграницы вступила в цифровую эру. Уже трудно найти в России университет, не занимающийся информационно-телекоммуникационными технологиями и не пытающийся применить их в учебном процессе. Многими учебными заведениями накоплен некоторый опыт разработки учебных материалов в электронном виде, ряд вузов даже начинает использовать электронное обучение в качестве дополнения к традиционному.

Таким образом, начинает формироваться новая сфера обучения, базирующаяся на знаниях в электронном виде и определяющая правила взаимодействия между ее участниками. В иностранной литературе она называется просто и емко — e-Learning.

Последней тенденцией в мировой практике является объединение большого числа организаций для создания консорциумов, перед которыми ставится задача разработки промышленных стандартов, эталонных моделей и открытых спецификаций, удовлетворяющих всем запросам среды. Объединение большого количества организаций-участников позволяет обеспечить финансовую сторону работы и привлечение к работе грамотных специалистов в востребованных областях знаний.

Все осложняется тем, что стандарты e-Learning разрабатываются в разных странах и важной задачей является их унификация, то есть разработка международных стандартов. Решением этой задачи занимается консорциум IMS, объединяющий разработчиков и пользователей систем электронного обучения.

Ключевым компонентом e-Learning является контроль и оценка знаний. Специально для этого консорциумом IMS разработан стандарт Question & Test Interoperability (IMS QTI).

На основании вышеизложенного можно сделать вывод, что разработка системы тестирования на основе международных стандартов, а именно IMS QTI, является актуальной на сегодняшний день.

Целью дипломной работы является разработка пилотной системы тестирования на основе стандарта IMS QTI (реализация хранения и предоставление тестовых заданий, на примере теста с выборочными ответами).

Для достижения данной цели необходимо решение следующих задач:

- изучение международного стандарта IMS QTI
- выбор технологий разработки системы тестирования
- проектирование модели системы тестирования на основе стандарта IMS QTI
- реализация пилотной системы тестирования на основе построенной модели

В качестве используемых технологий и стандартов выбраны: IMS QTI-Lite версия стандарта IMS QTI, промышленный стандарт XML, язык программирования Java, библиотека JDOM, Web-сервер Apache Tomcat.

Дипломная работа состоит из 4-х глав.

В первой главе «Спецификация IMS QTI» описаны ключевые элементы спецификации IMS QTI и их взаимосвязь.

Во второй главе «Выбор технологий и стандартов реализации» описываются технологии, выбранные для реализации проекта.

Третья глава «Описание системы тестирования» посвящена описанию общей структурной схемы системы с детализацией по пунктам отдельных ее составляющих.

В четвертой главе «Реализация системы тестирования» описывается практическая часть дипломной работы, представлены модули СТ, связанные непосредственно с тестированием.

1. Спецификация IMS QTI

Перед тем как проектировать систему тестирования, рассмотрим сначала саму спецификацию, она во много определит структуру будущей системы и используемые технологии. Основная задача данной главы — подробнее остановиться на спецификации IMS QTI, рассмотреть основные компоненты IMS QTI документов.

1.1 IMS Question and Test Interoperability

Одна из наиболее развитых спецификаций консорциума IMS (http://www.imsglobal.org) описывает структуры данных, используемые для обмена учебными материалами, предназначенными для оценивания результатов успеваемости обучаемых и информацией о результатах прохождения оценивания обучаемыми. Контроль полученных знаний один из основных элементов процесса обучения, поэтому спецификация достойна более широкого рассмотрения.

На данный момент доступна финальная версия 2.0 IMS QTI спецификации, которая была представлена в январе 2005 года и проходит апробацию версия QTI 2.1, которая доступна в виде общедоступного «черновика» (QTI 2.1 Public Draft Version 2). Спецификация пока не имеет завершенного характера, так, только в версии QTI 2.1 появился раздел, описывающий информационную модель для построения тестов.

Спецификация IMS QTI описывает модель данных используемых для представления вопросов (категория assessmentItem), тестов (категория assessmentTest) и для представления их результатов. Поэтому спецификация дает возможность обмена вопросов, тестов и итоговых данных между авторскими системами (authoring tools), базами данных вопросов (item banks), системами конструирования тестов (test constructional tools), системами обучения (learning systems) и системами взаимодействия и передачи данных (delivery system) [Рис 1.]

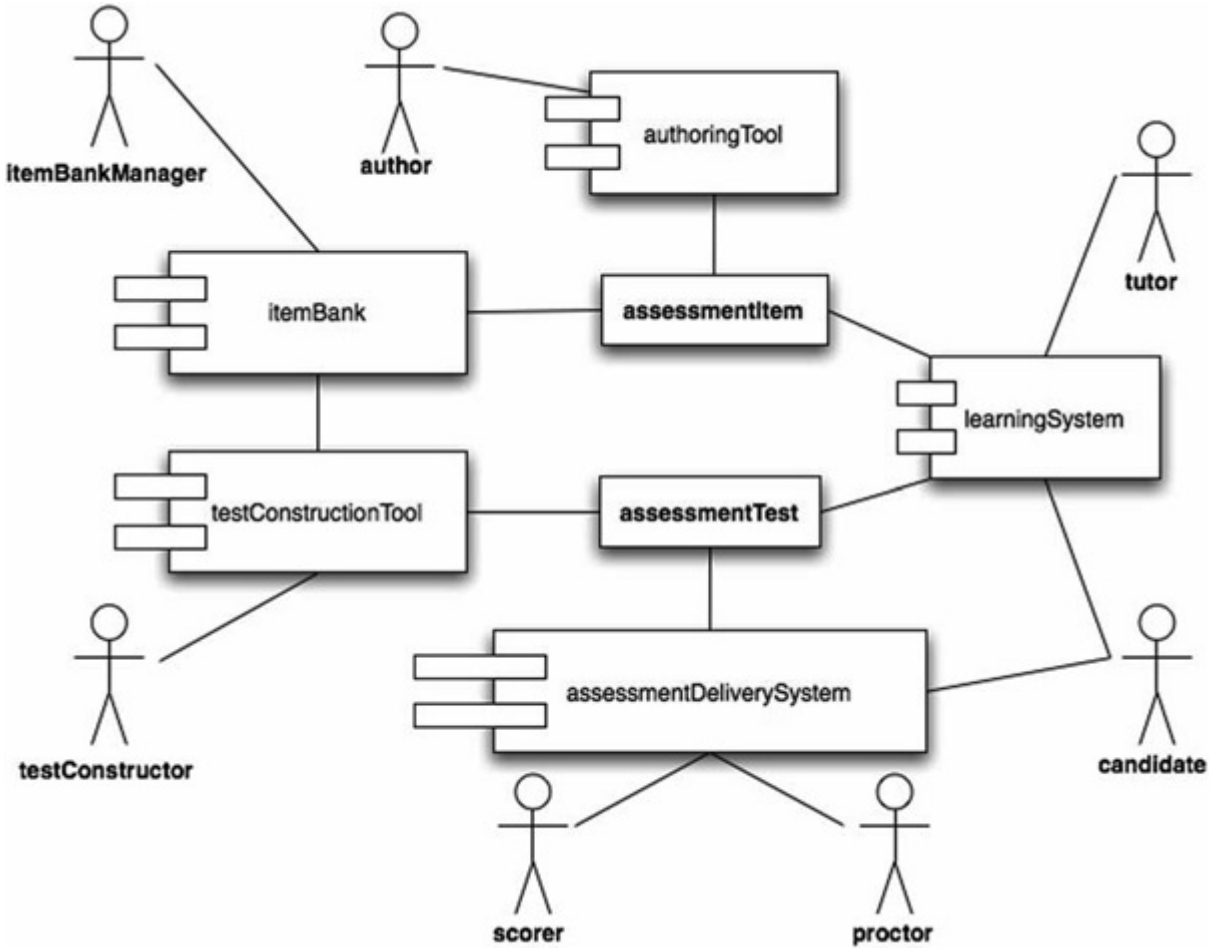

Рис.1

Схема системы тестирования, рекомендованная стандартом IMS QTI

Описание модели данных в спецификации носит абстрактный характер. Для этого используется UML (OMG Unified Modeling Language Specification), в виду большого разнообразия информационных моделей данных инструментальных сред и языков программирования. Тем не менее, для взаимообмена между системами предусмотрен промышленный стандарт XML (eXtensible Markup Language) и использование его сильно рекомендовано. Спецификация IMS QTI обеспечивает не только возможность взаимодействия, но и использование нововведений через строго определенные точки расширения [1].

Ознакомиться со спецификацией IMS QTI можно по адресу http://www.imsglobal.org/question/

1.2 Структура спецификации

Возьмем за основу финальную версию IMS QTI 2.0

Единая спецификация IMS QTI состоит из нескольких документов [1]:

- Implementation Guide — показывает реализацию информационной модели данных на примерах. Почти все примеры снабжены иллюстрациями. Хорошо подходит для читателей, впервые столкнувшихся с IMS QTI, показав, что же можно получить в итоге.
- Assessment Test, Section, and Item Information Model — справочное пособие об основной информационной модели данных вопросов (items) и тестов (tests). Подробная информация о модели данных, о требованиях к системе взаимодействия и передачи данных (delivery engines), авторским системам (authoring systems).
- Meta-data and Usage Data — описывает профиль IEEE Standard for Learning Object Metadata (LOM) модель данных, подходящая для использования в вопросах и тестах, и отдельная модель данных для представления рабочих данных (статистика вопросов). Этот документ представляет конкретный интерес для разработчиков и менеджеров баз данных вопросов и других репозиториев контента, и тем, кто создает тесты из этих хранилищ вопросов.
- Results Reporting — справочник о модели данных для представления результатов. Подробная информация о модели данных, о требованиях к системе взаимодействия и передачи данных (delivery engines)
- Integration Guide — описывает взаимоотношение между данной спецификацией и другими родственными спецификациями, например как IMS Content Packaging (IMS CP), IMS Simple Sequencing (IMS SS), and IMS Learning Design (IMS LD)
- XML Binding — описывает пути связывания информационной модели и формата XML

- Conformance Guide — описывает требования соответствия и обеспечения модели данных для конструирования профилей QTI включая предопределенный профиль, заместивший QTI Lite спецификацию, опубликованную как часть первой версии спецификации.
- Migration Guide — нацелен на людей, уже знакомых с версией 1.x. Описываются основные изменения, произошедшие в информационной модели данных, и включает алфавитный список элементов первой версии спецификации, снабженный подробной информацией о том, как эта информация представляется во второй версии.

Отметим несколько замечаний по терминологии.

Вопрос (Item) — комбинация опросного листа, предписания по визуальному отображению элементов вопроса (rendering), информации о правилах подсчета баллов за вопрос.

Секция (Section) — коллекция из 0 или более вопросов и/или секций

Тест (Test) — коллекция из 1 или более секций.

Спецификация описана абстрактно, состоит из описания классов, которые имеют атрибуты и могут включать другие классы, устанавливая соотношение родитель-ребенок.

В данной работе рассмотрены основные элементы спецификации: вопросы (item), секции (section) , тесты (test) и их основные составляющие.

1.3 Описание вопросов (items)

В данной спецификации термин вопрос (item) представляет собой широкое понятие и так как спецификация рассчитана на составление вопросов произвольной сложности, следовательно, объемлющий класс имеет множество атрибутов и подклассов на различных уровнях глубины. Сам элемент помимо формулировки вопроса и вариантов ответа, содержит информацию о верном ответе, правилах обработки выбранных вариантов ответов, возвращаемых переменных (как правило, оценка), информацию для обратной связи.

Объемлющим классом является класс **`assessmentItem`**, который имеет ряд атрибутов: `identifier, title, label, lang, adaptive, timeDependent, toolName, toolVersion.` Названия параметров говорят сами за себя, поясним только атрибут `identifier` — может возникнуть ситуация, когда имеется несколько элементов одной природы (не обязательно `assessmentItem`), а обрабатывать их надо по разным правилам, именно для этого необходим атрибут-идентификатор или же необходимо сопоставить связанные элементы разной природы (например, варианты ответов и правильный ответ). Типы данных спецификации можно узнать в тексте спецификации (Assessment Test, Section, and Item Information Model).

Наиболее значимые классы, которые включает класс `assessmentItem`

- `responseDeclaration` — описание переменных, связанных с ответами (правильный ответ).
- `outcomeDeclaration` — описание возвращаемых переменных (оценка)
- `itemBody` — представление вопроса: содержит текст, медиа-контент, элементы взаимодействия.
- `responseProcessing` — общие правила обработки ответа

Классы **responseDeclaration** и **outcomeDeclaration** являются представителями абстрактного класса `variableDeclaration`. Основные атрибуты: `identifier, cardinality, baseType`. Атрибут `cardinality` указывает на мощность множества значений переменной. Может быть одинарным (`single`) или множественным (`multiple, ordered, record`). Подкласс `defaultValue` указывает на значение по умолчанию.

Для класса `responseDeclaration` существует подкласс `correctResponse`, указывающий значение идентификатора правильного ответа.

Класс **itemBody** представляет собой контейнер, включающий текст, элементы взаимодействия (interactions). Разрешено использование определенных элементов формата XHTML (Extensible HyperText Markup Language), что позволяет более широко производить форматирование выводимой информации.

В зависимости от типа вопроса выбирают соответствующие элементы взаимодействия (interactions). Элементы взаимодействия по виду представления делятся на две группы `inlineInteraction` и `blockInteraction`, то есть размещаются прямой вставкой в текст вопроса или отдельным блоком.

К `inlineInteraction` относятся: `endAttemptInteraction, inlineChoiceInteraction, textEntryInteraction.`

К `blockInteraction` относятся: `associateInteraction, choiceInteraction, drawingInteraction, extendedTextInteraction, gapMatchInteraction, graphicInteraction, hottextInteraction, matchInteraction, orderInteraction, sliderInteraction, uploadInteraction.`

Наиболее распространенными и легкими в применении являются `choiceInteraction` (выбор ответа из приведенных вариантов) и `textEntryInteraction` (ввод ответа в строку ввода). Что из себя представляют остальные элементы взаимодействия можно посмотреть в разделе Implementation Guide спецификации IMS QTI. Перечислять атрибуты и подклассы для всех элементов взаимодействия нет необходимости, выбрав какой-то элемент взаимодействия, конкретно под него рассматриваются возможные атрибуты и подклассы.

Класс **`responseProcessing`** позволяет задать правила для обработки получаемого ответа. На основе подклассов выстраивается правило проверки (`responseRule`), состоящее из условий (`responseCondition`) и действий в зависимости от их выполнения (установка переменных, описанных в `outcomeDeclaration`). В качестве условий выступают выражения (expressions), наиболее распространенное — установка соответствия `match`.

Также поддерживается механизм шаблонов (template), путь к которому указывается в атрибуте `template` класса `responseProcessing`. Описываются шаблоны в соответствии с общими правилами. В спецификации описан ряд стандартных шаблонов. Для примера приведем стандартный шаблон `match_correct` (установка жесткого соответствия). Описание похоже на абстрактный язык программирования: есть условный оператор, который в зависимости от условия выполняет ту или иную ветвь. Общая схема обработка ответа представлена на Рис.2. В данном случае условием является строгое соответствие правильного ответа с полученным (оба имеют одинаковый идентификатор), в зависимости от условия устанавливается оценка 1 — все верно, иначе 0 [1].

Стандартный шаблон match_correct:

```
<responseProcessing>
      <responseCondition>
            <responseIf>
                  <match>
                        <variable identifier="RESPONSE"/>
                        <correct identifier="RESPONSE"/>
                  </match>
                  <setOutcomeValue identifier="SCORE">
                        <baseValue baseType="integer">1</baseValue>
                  </setOutcomeValue>
            </responseIf>
            <responseElse>
                  <setOutcomeValue identifier="SCORE">
                        <baseValue baseType="integer">0</baseValue>
                  </setOutcomeValue>
            </responseElse>
      </responseCondition>
</responseProcessing>
```

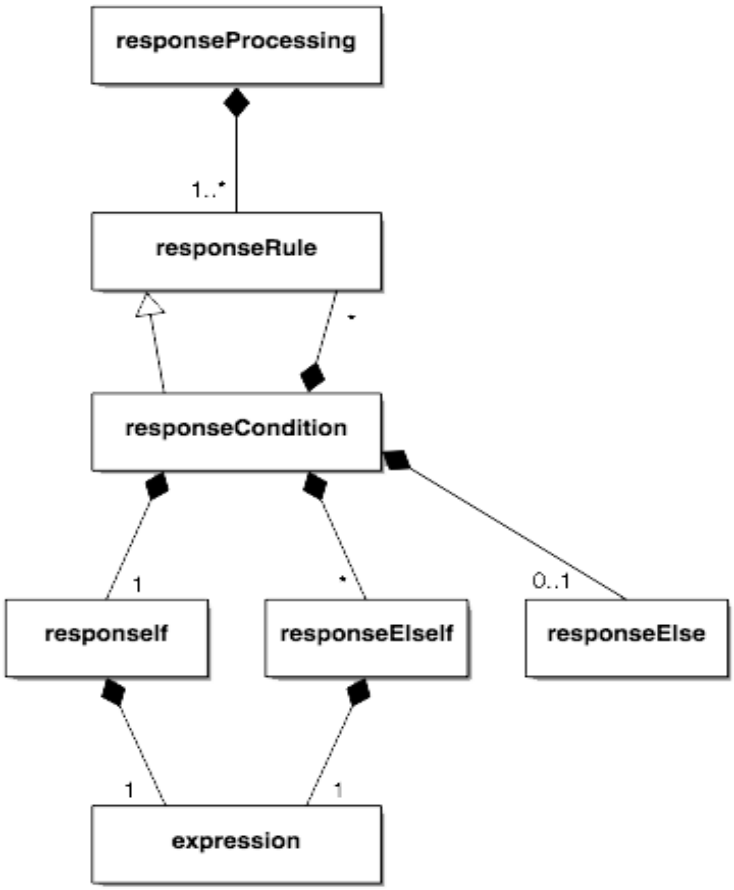

Рис.2

Схема обработка ответа в общем виде

Пример отдельного вопроса на базе выбора ответа из предложенных вариантов:

```
<?xml version="1.0" encoding="UTF-8"?>
<assessmentItem xmlns="http://www.imsglobal.org/xsd/imsqti_v2p1"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.imsglobal.org/xsd/imsqti_v2p1 imsqti_v2p1.xsd"
        identifier="choice" title="Unattended Luggage" adaptive="false" timeDependent="false">

        <responseDeclaration identifier="RESPONSE" cardinality="single" baseType="identifier">
                <correctResponse>
                        <value>ChoiceA</value>
                </correctResponse>
        </responseDeclaration>

        <outcomeDeclaration identifier="SCORE" cardinality="single" baseType="integer">
                <defaultValue>
                        <value>0</value>
                </defaultValue>
        </outcomeDeclaration>

        <itemBody>
                <p>Look at the text in the picture.</p>
                <p><img src="img/sign.png" alt="NEVER LEAVE LUGGAGE UNATTENDED"/></p>
                <choiceInteraction responseIdentifier="RESPONSE" shuffle="false" maxChoices="1">
                        <prompt>What does it say?</prompt>
                        <simpleChoice identifier="ChoiceA">
                                You must stay with your luggage at all times.
                        </simpleChoice>
                        <simpleChoice identifier="ChoiceB">
                                Do not let someone else look after your luggage.
                        </simpleChoice>
                        <simpleChoice identifier="ChoiceC">
                                Remember your luggage when you leave.
                        </simpleChoice>
                </choiceInteraction>
        </itemBody>

        <responseProcessing
                template="http://www.imsglobal.org/question/qti_v2p0/rptemplates/match_correct"/>
</assessmentItem>
```

1.4 Описание секций (sections) и тестов (tests)

Описание модели построения секций и тестов появилась только в версии 2.1 спецификации IMS QTI, которая носит пока что статус «черновика», поэтому рассмотрим только общие механизмы объединения отдельных вопросов в секции, а секций в тесты.

Тест — это группа вопросов `assessmentItems`, объединенные в секции, и связанные с ней инструкции, которые определяют какие элементы видимы, в каком порядке они представлены и каким образом происходит взаимодействие.

Объемлющим классом при описании тестов является **`assessmentTest`**, который имеет такие же атрибуты как и `assessmentItem`, но включает такие классы как:

- `timeLimits` — определяет ограничение по времени
- `testPart` — основное разделение теста на части (части состоят из секций, секции из вопросов; характеристики наследуются).
- `outcomeProcessing` — обработка полученных переменных (оценок)

Класс **`testPart`** как основной делитель теста на части, представляет механизмы контроля выполнением теста, которые определяются как атрибуты:

- `navigationMode` — может принимать два значения: `linear`, то есть тест будет выполняться в линейном порядке без права возвращения к пройденным вопросам, и `nonlinear`, то есть ограничение на перемещение снимается, в любое время доступен любой вопрос.
- `submissionMode` — определяет, когда выбранные ответы будут передаваться на обработку. Может принимать два значения:

individual, то есть требование предоставлять данные на обработку с каждого вопроса, и simultaneous, то есть возможные ответы передаются все вместе по окончанию части теста (testPart).

Части теста состоят из секций, которые описываются классом **assessmentSection**, который позволяет управлять отдельными вопросами. Так из всех вопросов, используя класс selection, можно предоставить к прохождению только определенное число вопросов, а используя атрибут withReplacement этого класса, можно задать перетасовку вопросов; используя класс ordering, можно задать порядок вопросов для каждой части теста.

Для включения вопроса в секцию используется класс assessmentItemRef, который своими атрибутами identifier и href задает идентификатор и путь до файла с вопросом; посредством класса weight можно задать «вес» вопроса (сложность) [1].

Пример объединения вопросов в тест.

```
<?xml version="1.0" encoding="UTF-8"?>

<assessmentTest xmlns="http://www.imsglobal.org/xsd/imsqti_v2p1"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.imsglobal.org/xsd/imsqti_v2p1
        http://www.imsglobal.org/xsd/imsqti_v2p1.xsd"
        identifier="RTEST-01" title="Sets of Items WithLeading Material">

        <testPart identifier="P1" navigationMode="linear" submissionMode="individual">
                <assessmentSection identifier="set" title="Section A" visible="true">
                        <selection select="2"/>
                        <assessmentItemRef identifier="set01" href="rtest01-set01.xml"/>
                        <assessmentItemRef identifier="set02" href="rtest01-set02.xml"/>
                        <assessmentItemRef identifier="set03" href="rtest01-set03.xml"/>
                        </assessmentSection>
        </testPart>
</assessmentTest>
```

2. Выбор технологий и стандартов реализации

Отправной точкой в цепочке используемых технологий является рекомендация консорциумом IMS использования промышленного стандарта XML (eXtensible Markup Language) при обмене учебными данными. Рассмотрим более подробно всю цепочку.

2.1 Профиль IMS QTI-Lite

Так как реализовать все возможности спецификации IMS QTI крайне затруднительно, да и нет такой необходимости, был выбран профиль IMS QTI-Lite, который является подмножеством спецификации IMS QTI и характеризуется [1]:

- только один элемент взаимодействия в вопросе
- поддерживаемые типы элементов взаимодействия: `choiceInteraction` (используется для создания тестов с выборочными ответами, такие как выбор одного ответа из многих, выбор одного или более ответа из многих)
- использование простых правил обработки ответов на базе стандартного шаблона match correct (строгое соответствие).
- нет поддержки встроенной обратной связи
- ограничение типов изображений и структурных элементов форматирования
- нет поддержки расширенных возможностей, например адаптивные вопросы, шаблоны, ограничение по времени.

Полный список разрешенных и запрещенных опций можно посмотреть на сайте проекта:
http://www.imsglobal.org/question/qtiv2p1pd2/conformance/imsqti_lite_profile.xml

Использование профиля IMS QTI-Lite позволяет создавать, используя элемент взаимодействия `choiceInteraction`, наиболее распространенный тип вопросов — вопросов с выборочным ответом.

По спецификации класс `choiceInteraction` относиться к классу `blockInteraction`, от которого наследует подкласс `prompt`, отвечающий за формулировку вопроса; класс `blockInteraction` в свою очередь наследует от общего класса `Interaction` атрибут `responseIdentifier`, который служит для связи с `responseDeclaration` и, соответственно с идентификаторами правильных ответов `correctResponse`.

Сам класс `choiceInteraction` имеет атрибуты:

- `shuffle : boolean` — случайный вывод вариантов ответов, по умолчанию `false`
- `maxChoices : integer` — максимальное число ответов, которые может дать пользователь, по умолчанию 1, если равно 0, то игнорируется.

и содержит от 1 и более подклассов `simpleChoice`, отвечающих за содержание вариантов ответа.

Хранить и передавать QTI данные в системе тестирования, согласно спецификации, будем в виде XML документов.

2.2 Стандарт XML

XML [http://www.xml.org, http://www.xml.com] (eXtensible Markup Language — расширяемый язык разметки) — это стандартный, независимый от системы способ представления данных. Как и HTML (HyperText Markup Language — язык разметки гипертекста), XML заключает данные внутрь тэгов, однако между этими языками существуют

значительные различия. Во-первых, тэги XML связаны с содержанием текста, заключенного между ними, в то время как тэги HTML определяют способ отображения текста. Следующий пример отображает прайс-лист с названием и ценой двух сортов кофе:

```
<priceList>
      <coffee>
            <name>Mocha Java</name>
            <price>11.95</price>
      </coffee>
      <coffee>
            <name>Sumatra</name>
            <price>12.50</price>
      </coffee>
</priceList>
```

Тэги `<coffee>` and `</coffee>` указывают синтаксическому анализатору, что информация между ними касается кофе. Два других тэга внутри тэгов `<coffee>` определяют, что заключенная в них информация — это название кофе и его цена за 1 фунт. Благодаря тому, что XML тэги показывают содержимое и структуру данных, заключенных в них, они делают возможными такие операции, как архивирование и поиск.

Второе значительное отличие между HTML и XML состоит в том, что тэги XML являются расширяемыми, что позволяет писать собственные тэги для описания нужного содержания. С HTML вы ограничены тем набором тэгов, которые были предопределены спецификацией HTML.

Благодаря расширяемости, которую предоставляет XML, вы можете создавать тэги, которые вам требуются для данного конкретного типа документа [6].

Так как основной поток данных будет в виде XML файлов, которые в процессе работы системы тестирования будут генерироваться, анализироваться, модифицироваться, то для этого необходим довольно мощный инструментарий.

2.3 Язык программирования Java

Среди средств разработки приложений способных работать с QTI XML документами, можно выделить язык программирования Java. Одна из черт, сближающих Java и XML, — то, что при разработке обоих средств приоритет был отдан удобству пользователей и разработчиков, нежели технологическим интересам. Другими словами, при реализации и Java, и XML создателей более заботило удобство работы пользователей, нежели скорость выполнения программ или оптимизация под какой-либо тип компьютерной платформы. В результате одной из главных отличительных особенностей Java стал интерпретируемый код: написано однажды, работает везде. В XML же был реализован принцип записи всей информации в текстовом формате, что гарантирует удобство пользователей и независимость от оборудования и программного обеспечения.

Java имеет мощные API (Application Program Interface) для работы с XML документами. На сегодняшний момент выделяют две основные модели анализа XML документов: анализ, основанный на событиях (event-based parsing), и анализ, основанный на структуре дерева документа (tree-based parsing).

SAX (Simple API for XML) создает прикладной программный интерфейс для синтаксического анализатора, основанного на событиях. Основанный на событиях анализатор просматривает XML документ от начала и до конца, посылая сообщение запущенному приложению каждый раз, когда он встречает синтаксическую конструкцию.

DOM API (Document Object Model — объектная модель документа) представляет собой набор интерфейсов для построения объектного представления в форме дерева анализируемого XML документа. Построив DOM один раз, вы можете управлять ею с помощью DOM-методов, таких как insert и remove (вставка и удаление), абсолютно аналогично работе с

любыми другими деревьями. Таким образом, в отличие от синтаксического анализатора SAX, анализатор DOM позволяет произвольный доступ к каким-либо конкретным частям документа. Другое отличие состоит в том, что с помощью SAX можно только считывать XML документы, а DOM дает вам возможность построить в памяти объектную модель и работать уже с ней, добавляя новые элементы и удаляя ненужные [8].

Кроме того, существуют сторонние библиотеки, которые позволяют еще более усилить связку XML — Java.

2.4 JDOM API

Во многих отношениях язык Java стал языком программирования для XML. Но хотя множество Java-разработчиков используют XML ежедневно, Sun задерживает индустриальное внедрение XML в платформу Java. Поскольку платформа Java 2 прекрасно развивалась до того, как XML обозначил себя как ключевая технология для всего, от интеграции бизнеса с бизнесом и до наполнения Web-сайтов. Наиболее значительным на сегодня добавлением было введение JAXP (Java API for XML Parsing). Чего недостает в базовой платформе Java, так это интуитивного интерфейса для манипулирования XML документами как Java-объектами.

С появлением JDOM ситуация меняется. JDOM является уникальным Java-инструментом для работы с XML, он создан для обеспечения быстрой разработки XML-приложений. В его проекте использованы синтаксис и семантика языка Java.

JDOM был разработан и введен в действие как проект с открытым кодом по лицензии, подобной Apache, в начале 2000 года. Цель проекта — построить полное решение на платформе Java для обращения, манипулирования и вывода XML-данных из кода Java. В настоящий

момент разработана финальная версия JDOM 1.0, познакомиться с которой можно на сайте проекта http://jdom.org.

JDOM может использоваться как альтернатива пакету org.w3c.dom для программного манипулирования XML документами. Это не обязательная замена, фактически JDOM и DOM могут счастливо сосуществовать. Кроме того, JDOM не занимается разбором исходного текста XML, хотя он обеспечивает классы-оболочки, которые берут на себя большую часть работы по конфигурированию и выполнению реализации анализатора. JDOM использует сильные стороны существующих API.

Кроме того JDOM избавлен от многих недостатков и ограничений модели DOM таких как:

- DOM не была разработана под конкретный язык, хотя такой подход и помогает построить похожие API под разнообразные языки, но в тоже время делает их более громоздкими. Также порождается избыточность классов. Например, хотя язык Java имеет встроенный в язык класс `String`, спецификация DOM определяет собственный класс `Text`.
- API DOM следует непосредственно самой спецификации XML. В XML все является узлами (node), в итоге DOM — интерфейс на базе узлов, где почти все строится на методах, которые возвращают `Node`. Это удобно с точки зрения полиморфизма, но порождает громоздкий и плохо понимаемый код.
- DOM API состоит только из интерфейсов. Для определенных вариантов использования, когда XML документы строятся только анализатором и никогда — прикладным кодом, это не имеет значения. Но когда использование XML становится более широким, не все проблемы продолжают оставаться управляемыми только анализатором, и разработчики приложений нуждаются в удобном способе конструировать объекты XML программно.

Напротив, JDOM сформулирован как легкий API, прежде всего ориентированный на интеграцию с Java и имеет свои особенности:

- JDOM специализирован для платформы Java. API использует, где возможно, встроенную в Java поддержку `String`. Он также использует классы коллекций платформы Java 2, такие как `List` и `Iterator`, обеспечивая богатую среду для работы программистов, хорошо знакомых с языком Java.
- Нет иерархий. В JDOM элемент XML является экземпляром класса `Element`, атрибут XML является экземпляром класса `Attribute`, а сам XML документ является экземпляром класса `Document`. Поскольку все они представляют разные концепции в XML, они всегда представляются собственными типами, а не как аморфные «узлы».
- Поскольку объекты JDOM являются непосредственными экземплярами таких классов, как `Document`, `Element` и `Attribute`, создание их настолько легко, насколько легко использование оператора `new` языка Java. Это также означает, что нет необходимости в интерфейсе фабрики для конфигурирования — JDOM готов к использованию прямо из jar.

Использование JDOM намного упрощает программирование XML. Для примера рассмотрим построение простого XML документа:

```
<?xml version="1.0" encoding="UTF-8"?>
<car vin="123fhg5869705iop90">
  <make>Toyota</make>
  <model>Celica</model>
  <year>1997</year>
  <color>green</color>
</car>
```

Сначала надо создать документ и добавить в него корневой элемент:

```
Element carElement = new Element("car");
Document myDocument = new Document(carElement);
// так как корневой элемент в документе всегда 1, то
// Document принимает в конструкторе Element
```

Далее добавляем атрибут vin:

```
carElement.addAttribute(new Attribute("vin", "123fhg5869705iop90"));
```

Создаем и добавляем подэлемент:

```
Element make = new Element("make");
make.addContent("Toyota");
carElement.addContent(make);
```

Поскольку метод `addContent` класса `Element` возвращает `Element`, мы можем добавлять элементы в краткой форме:

```
carElement.addContent(new Element("model").addContent("Celica"));
<…>
```

Манипуляция элементами документа осуществляется похожим образом, например:

Обращение к дочерним элементам:

```
Element yearElement = carElement.getChild("year");
// вернет элемент year
List itemList = carElement.getChildren();
// вернет список всех дочерних элементов
```

Удаление дочерних элементов:

```
boolean removed = carElement.removeChild("year");
// удалит только элемент year
```

Получить экземпляр документа можно и из существующего XML документа (далее работы с ним аналогична):

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(new File("/some/directory/sample.xml"));
```

Класс `XMLOutputter` позволяет вывести имеющийся экземпляр документа:

```
XMLOutputter outputter = new XMLOutputter("  ", true);
// два пробела отступа по иерархии и перевод строки после элемента
FileWriter writer = new FileWriter("/some/directory/sample.xml");
outputter.output(myDocument, writer);
writer.close();
```

При указании пространства имен, получив его:

```
Namespace ns = root.getNamespace();
```

его можно использовать в дальнейшем, указывая в числе параметров [5].

Кроме стандартного манипулирования XML документами, в JDOM реализован язык XPath [http://www.w3.org/TR/xpath] (XML Path Language) — язык для обращения к частям XML документа. Его использование намного сокращает код при поиске элемента, обладающего некими заданными характеристиками (особенно при наличии большого числа одноименных элементов).

XML документ обладает древовидной структурой и XPath призван помочь обходить всевозможные деревья, получать необходимые элементы из другой ветви относительно точки обхода, узнавать родителей, детей, атрибуты. Это полноценный язык навигации по дереву.

Для нахождения элементов в дереве документа используются пути адресации. Рассмотрим путь адресации /html/body/div[@class] (или /child::html/child::body/child::div[attribute::class] в полном виде). Косыми чертами отделяются шаги адресации, каждый из которых состоит из трех частей:

- ось (в данном примере child::), это обязательная часть;
- условие проверки узлов (в данном примере это имена элементов документа html, body, div, а символ * означает элемент с любым именем), это обязательная часть;
- предикат (в данном примере attribute::class), необязательная часть заключаемая в квадратные скобки, в которой могут содержаться оси, условия проверки, функции, операторы (+, -, <, > и проч.).

Путь адресации анализируется слева направо, на каждом шаге адресации перебираются все дочерние элементы, устанавливается соответствие имен и исчисляются предикаты, в итоге анализатор вернет элементы div (у которых предком является body и т.д. по иерархии пути) имеющие атрибут class.

Рассматривать все оси XPath не будем, ограничимся указанием наиболее важных и их сокращений [9]:

- attribute:: — возвращает множество атрибутов текущего элемента; можно заменить на @
- child:: — возвращает множество потомков на один уровень ниже; часто просто опускают
- descendant:: — возвращает полное множество потомков; можно заменить на //
- parent:: — возвращает предка на один уровень назад; можно заменить на ..
- self:: — возвращает текущий элемент; можно заменить на .

2.5 Web-сервер Apache Tomcat

Разработка системы тестирования предполагает расположение ее на выделенном компьютере, будь то локальная или глобальная сеть. Наиболее удобным способом обмена данными в сети представляется протокол HTTP, потому что тогда не нужно будет заботиться о стороне клиента, так как web-браузер есть почти на каждом компьютере. Таким образом, задачей системы тестирования на данном этапе является прием и обработка запросов клиента, и ответ на них — генерация HTML страниц с релевантным контентом.

Поэтому язык программирования Java был выбран не только как отличное средство для манипулирования XML документами, но и потому что существуют Java-спецификации JSP (Java Server Pages) и Java Servlets, расширяющие возможности web-серверов.

Java Servlet (или просто сервлет) — это класс, который расширяет абстрактный класс javax.servlet.http.HttpServlet и динамически генерирует контент в ответ на HTTP запросы клиентов. Web-сервера, поддерживающие сервлеты, при поступлении запроса вызывают

соответствующие методы сервлета, методам передаются все необходимые параметры для получения информации о запросе и формирования ответа.

JSP (JavaServer Pages) — технология, позволяющая динамически генерировать web-страницы, внедрять Java-код, а также EL (expression language) в статичное содержимое страницы. Страницы транслируются в Java-код сервлета с помощью компилятора JSP страниц Jasper, и затем компилируется в байт-код виртуальной машины Java [6].

Web-сервера, которые поддерживают работу сервлетов, называют сервлет-контейнерами. Существует множество web-серверов, поддерживающих сервлеты. Наиболее популярный — это Apache Tomcat [http://tomcat.apache.org/]. Tomcat может работать и как отдельное приложение или интегрироваться в Apache HTTP Server в качестве модуля.

По умолчанию, Tomcat работает на порту 8080, который можно изменить в файле conf\server.xml. Каталог webapps является корневым каталогом для размещения веб-сайтов.

На июнь 2007 года самая последняя версия Tomcat 6.0. Эта версия поддерживает спецификации Java Servlet 2.5 и JavaServer Pages 2.1.

3. Описание системы тестирования

Основная задача данной главы — описать общую структуру разрабатываемой системы тестирования (СТ) и ее функционирование, а так же структуру отдельных модулей и их взаимодействие.

3.1 Структурная схема СТ и ее функционирование

Схема работы системы тестирования приведена в Приложении 1. Основными частями системы являются:

- Базы данных (БД)
- Программные модули
- Раздельный интерфейс для работы с системой пользователей, имеющих в ней различный статус (учитель, ученик)

В системе вся необходимая информация хранится в трех базах данных:

- БД информации о пользователях
- БД информации о существующих в системе группах пользователей
- БД предметных тестов

Так как основной поток данных будет в виде QTI XML файлов и для манипулирования XML документами будут написаны соответствующие методы, то было решено основные базы данных в системе реализовать в виде файловой БД XML документов. Подробнее о структуре БД в пункте 3.2

Программных модулей в системе шесть:

- Авторизация / регистрация
- Тестирование, включающий:
    - плеер тестов

    - запись результата и процесса прохождения теста
- Менеджер тестов, включающий
    - конструктор тестов
    - редактирование тестов
    - удаление тестов
- Менеджер групп, включающий:
    - создание новой группы
    - редактирование существующей группы
- Просмотр результатов
- Менеджер пользователей, включающий:
    - Удаление пользователя
    - Редактирование информации о пользователе (изменение статуса, принадлежность к группе, список назначенных тестов)

Для работы с системой учеников и учителей реализованы два различных интерфейса. Переход к ним осуществляется при авторизации. При загрузке основной страницы системы пользователю предлагается ввести логин, пароль и свой статус (так как пользователь может иметь несколько статусов), после этого на стороне сервера выполняется модуль авторизации, и в зависимости от успешности авторизации и статуса пользователя, выводится соответствующий интерфейс:

- Учитель: имеет возможность создавать, редактировать, удалять, назначать тесты; создавать, редактировать (менять состав) группы; редактировать профили учеников, просматривать результаты тестирования учеников.
- Ученик: имеет возможность пройти тест, назначенный ему учителем, и просмотреть свои результат пройденных тестов.

3.2 Структура баз данных

Принято решение исполнять базы данных в виде XML документов, но это приносит некоторые трудности в организации БД. Так, например, если создавать БД единым файлом и достигать иерархичности записей путем вложенных элементов, то при большом количестве пользователей, эти XML файлы будут достигать больших размеров, что непременно скажется на скорости их обработки, занимаемой при этом памяти и, следовательно, быстродействии системы в целом. Поэтому там где это возможно и целесообразно иерархия будет достигаться путем использования директорий и поддиректорий файловой системы, в которых будут располагаться XML документы с соответствующим контентом.

<u>База данных информации о пользователях:</u>

Организуется за счет общего для всех пользователей файла users.xml и дерева папок

```
/users
│
└──<идентификатор пользователя>
   │   results.xml
   │   appt.xml
   │
   └──results
        <идентификатор результата>.xml
        <…>
```

то есть в корневой папке системы, есть папка `users`, которая содержит папки под каждого отдельного пользователя (создаются при регистрации), в качестве имен папок служат уникальные идентификаторы, выданные пользователям при регистрации.

В папке каждого пользователя при регистрации создаются файлы appt.xml (список назначенных тестов) и results.xml (список пройденных тестов).

Так же внутри папки пользователя создается папка results, в которой хранится информация о пройденных тестах, каждый отчет в отдельном файле (имена — уникальные идентификаторы выдаваемые системой при прохождении теста, контент — согласно спецификации IMS QTI).

Сам файл users.xml имеет вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<users>

  <last>2</last>

  <role rolename="учитель" />
  <role rolename="администратор" />
  <role rolename="студент" />

  <user login="Смирнов А.М." passhash="B26986CEEE60F744534AAAB928CC12DF"
              roles="студент" groups="2" id="1" />

  <user login="tutor" passhash="A8F5F167F44F4964E6C998DEE827110C"
              roles="студент,учитель" groups="none" id="2" />
  <…>

</users>
```

Тег `<last/>` используется для формирования уникального идентификатора пользователя, путем простого инкремента его значения. Теги `<role/>` перечисляют статусы пользователей, существующие в системе. Тег `<user/>` отвечает за хранение информации о пользователе и имеет ряд атрибутов:

- login — логин пользователя в системе
- passhash — md5-хэш от пароля (то есть пароль в открытом виде в системе не хранится)
- roles — какими статусами обладает пользователь (по умолчанию «студент»)
- groups — к каким группам принадлежит (указывается идентификатор группы, по умолчанию «none»)
- id — уникальный идентификатор пользователя

Файл appt.xml имеет вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<appt>
    <item title="Алгоритмизация: Циклы (8-10)">12</item>
</appt>
```

Тег <item/> отвечает за один назначенный тест, атрибутом указывается заголовок теста, содержащееся значение — идентификатор теста в системе.

Файл results.xml имеет вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<results>

  <last>2</last>

  <test resid="2" testid="12" title="Алгоритмизация: Циклы (8-10)"
                                                  date="20070606T153948" />
</results>
```

Аналогично, тег <last/> используется для формирования уникального идентификатора результата, путем просто инкремента его значения. Тег <test/> используется для хранения информации о результате пройденного теста и имеет ряд атрибутов:

- resid — идентификатор результирующего отчета
- testid — идентификатор выполненного теста
- title — заголовок теста
- date — дата и время прохождения теста в формате ISO8601

БД групп пользователей:

В системе тестирования реализована система групп пользователей (будь то классы, параллели, в общем, группы пользователей по какому-либо признаку деления). Существует общая группа «none», к которой пользователи относятся при регистрации, то есть изначально пользователи не относятся к какой-либо другой группе. Членство пользователя в группах определяется учителем.

Организуется за счет общего для всех пользователей файла

groups.xml и отдельных XML документов на каждую группу в корневой папке groups.

```
/groups
|      none.xml
|      <идентификатор группы>.xml
|      <…>
└─
```

Файл none.xml имеет вид (остальные аналогично):

```
<?xml version="1.0" encoding="UTF-8"?>
<group>
  <member login="tutor">4</member>
  <member login="newuser">19</member>
  <member login="Смирнов А.М.">20</member>
  <member login="Иванов П.К">1</member>
</group>
```

Тег `<member/>` отвечает за хранение информации о члене группы, имея в качестве атрибута логин пользователя, а в качестве значения его идентификатор.

При удалении пользователя из всех созданных групп, он автоматически переходит в общую группу «none».

БД предметных тестов:

Организуется за счет общего для всех пользователей файла tests.xml и отдельных директорий в корневой папке tests под каждый тест, в котором хранятся QTI XML документы, соответственно, тест (test.xml) и вопросы к нему (choiceN.xml).

```
/tests
|
└──<идентификатор теста>
     |   test.xml
     |   choice1.xml
     |   <…>
     |   choiceN.xml
     └──
```

Файл tests.xml имеет вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<tests>
  <last>1</last>
  <test id="1" theme="Алгоритмизация" title="Циклы" classrange="8-10" />
</tests>
```

Тег `<last/>` используется для формирования уникального идентификатора теста, путем просто инкремента его значения. Тег `<test/>` отвечает за хранение данных о тесте и имеет ряд атрибутов:

- id — уникальный идентификатор теста в СТ
- theme — глобальная тема теста
- title — конкретная тема теста
- classrange — указание какой категории учеников можно назначать тест

4. Реализация системы тестирования

Основная задача этой главы — прокомментировать практическую часть дипломной работы, то есть саму реализацию системы тестирования, описать реальные воплощения основных модулей системы, связанных непосредственно с тестированием, и механизмов взаимодействия узлов структурной схемы системы.

4.1 О работе системы в целом.

При реализации системы тестирования основные узлы структурной схемы [Приложение 1] получили исполнение в виде:

- Программный модуль — сервлет
- Интерфейс — JSP страница или динамически генерируемая сервлетом HTML страница
- База данных — файловая база данных на основе XML документов

Система тестирования состоит из:

7 сервлетов:

- login — авторизация пользователя
- register — регистрация пользователя
- player — воспроизведение тестов, запись результатов
- replayer — воспроизведение результатов
- users — менеджер пользователей
- tests — менеджер тестов
- groups — менеджер групп

3 вспомогательных класса агрегирующие однородную информацию:

- data_item — информация о вопросе
- data_test — общая информация о тесте

- data — полная информация о выполняемом тесте

4 JSP страницы:

- index.jsp — начальная страница (авторизация или перенаправление на страницу, соответствующую статусу)
- register.jsp — форма регистрации нового пользователя
- student.jsp — интерфейс работы с системой ученика
- tutor.jsp — интерфейс работы с системой учителя

3 основных XML файла БД (полная структура в пункте 3.2):

- users.xml — список пользователей, зарегистрированных в системе тестирования
- tests.xml — список тестов, имеющихся в СТ
- groups.xml — список групп, действующих в СТ

Кроме того, создан вспомогательный класс _tools включающий:

- текст системных сообщений
- группы функций:
    - облегчающие работу с XML документами
    - работы с паролями
    - работы с датой и временем
    - отображающие системные сообщения
    - реализующие web-интерфейс
    - общие для нескольких модулей системы
    - определяющие принадлежность и эквивалентность элементов друг другу

Реализация интерфейсов и программных модулей системы тестирования в виде JSP страниц и сервлетов определила механизм их

взаимодействия — HTTP запросы (GET и POST). Методы GET и POST различаются способом передачи параметров (GET — передача параметров в URL, POST — в теле запроса). Все данные из форм в системе тестирования отправляются методом POST, а так как по умолчанию данные отправляются методом GET, то запрос к модулям по ссылкам будет передаваться методом GET.

При получении запроса, сервлет-контейнер (Tomcat) вызывает у соответствующего сервлета метод doPost(), если пришел POST запрос, или doGet(), если пришел GET запрос. Это поможет разграничить обработку запросов от форм и запросов по ссылкам.

Запросы POST и GET являются транспортом для запросов системы тестирования. Взаимодействие узлов системы реализуется за счет передачи и последующей обработки параметров через HTTP запросы. Методам doPost() и doGet() передаются два параметра:

- `HttpServletRequest request` — информация о запросе
- `HttpServletResponse response` — объект для формирования ответа

Из информации о запросе можно узнать значение передаваемых параметров по их имени:

```
String cmd = request.getParameter("cmd");
//узнаем значение параметра cmd
```

Помимо передачи параметров, взаимодействие между узлами системы тестирования организовано за счет механизма HTTP сессий — для каждого клиента формируется уникальный идентификатор, который передается с каждым запросом к серверу, что позволяет идентифицировать его среди всех остальных клиентов. Сервер для каждой сессии хранит переменные сеанса, добавляемые и изменяемые клиентом.

Через механизм сессий в системе тестирования организована авторизация и последующее отслеживание статуса клиента (разграничение

прав доступа), передача вспомогательных данных (например, классов data, data_item, data_test; переменных идентификации: логин, роль, идентификатор). Данные сессии сохраняются на все время сеанса работы или до того момента как их удалят из сессии.

Перед получение или изменением данных (атрибуты сессии), сохраненных в сессии, нужно получить из запроса экземпляр класса HTTPSession текущей сессии:

```
HttpSession session = request.getSession();
```

Для манипулирования данными определен ряд методов:

- `Object getAttribute(String name)` — получение объекта атрибута сессии с именем name
- `void setAttribute(String name, Object value)` — установка атрибута сессии с именем name объектом value
- `void removeAttribute(String name)` — удаление атрибута с именем name из сессии

Например:

```
String id = (String) session.getAttribute("id");
List appt = (List) session.getAttribute("profile_appt");
// получение атрибута по имени
// необходимо приведение типов

session.setAttribute("login", "Иванов");
// установка атрибута с именем "login" значением "Иванов"
```

Передача параметров через HTTP запросы и данных через сессию вместе образуют единую систему взаимодействия узлов системы тестирования.

4.2 Тестирование

Реализовано в сервлете player, который совмещает функции воспроизведения теста и сохранения результата тестирования (итог и процесс прохождения теста).

Работа сервлета заключается в анализировании QTI XML файлов теста и вопросов к нему, отображения вопросов, обработки ответов ученика, запись результата тестирования. При работе сервлета активно используется класс data общей информации о тесте и вопросах и глобальные переменные самого сервлета:

```
import java.util.Vector;
public class data {
        String testFileName = "test.xml";
        String testTitle = "";
        int questionsCount = 0; // количество вопросов
        int Score = 0; // итоговый результат
        Vector itemsHref = new Vector();// имена файлов отдельных вопросов
        Vector itemsID = new Vector(); // список ID вопросов
        Vector scores = new Vector(); // список результатов за вопросы

        String topFrame = ""; // xhtml-контент перед выбором ответа
        String bottomFrame = ""; // xhtml-контент после выбора ответа
        String feedback= ""; // обратная связь
        String prompt = ""; // вопрос
        boolean Multiple = false; // выбор более 1 варианта   ответа
        boolean Single = true; // выбор только одного варианта ответа
        int maxChoices = 1; // макс число вариантов выбора.
        int choicesCount = 0; // количество вариантов ответов в вопросе
        int answersCount = 0; // количество правильных ответов
        int currentQuestion = 0; // номер текущего вопроса
        int itemScore = 0; // результат за вопрос

        // список идентификаторов правильных ответов
        Vector answersID = new Vector();
        Vector choices = new Vector(); // список вариантов ответов
        // список идентификаторов вариантов ответов
        Vector choicesID = new Vector();
        Vector choicesResponse = new Vector(); // список выбранных ответов
}

public class player extends HttpServlet {
        data tt; // информация о тесте и вопросах
        int current; // текущий вопрос
        int i;

        _tools f = new _tools(); // класс вспомогательных функций
        String context; // контекст сервлета
        PrintWriter out; // поток вывода
        String appPath; // путь к папке проекта
        HttpSession session; // класс http-сессии
        String role; // роль(статус) пользователя
        String id; // идентификатор пользователя
```

```
        boolean isTopFrame; // обработка верхнего заголовка
        String tagName; // имя тега

        // переменные для записи результата
        static Document docRes;
        static Element itemResult;
        static Element candidateResponse;
        static Element testResult;
        static Namespace nmRes;

        static int resID; // идентификатор результата
        <…>
}
```

Выполнение теста начинается с нажатия на ссылку вида /test/player?testid=46 в интерфейсной части системы (выполнение назначенного теста учеником, просмотр теста учителем). Соответственно параметр testid (уникальный идентификатор теста в системе) передается GET запросом и сервлет-контейнер вызывает метод doGet() сервлета player, передавая ему информацию о запросе и объект для формирования ответа.

Метод doGet() вызывает метод инициализации глобальных переменных init(), передавая ему параметры запроса, и вызывает основной модуль обработки doModule().

```
// инициализация глобальных переменных
// и установка соотвествующих параметров ввода / вывода
private void init(HttpServletRequest req, HttpServletResponse res) throws
                                                          IOException {
      // устанавливаем формат ответа
      res.setContentType("text/html; charset=UTF-8");
      out = new PrintWriter(new OutputStreamWriter(res.getOutputStream(),
                           "UTF8"), true); // получаем поток вывода
      context = req.getContextPath(); // получаем контекст сервлета
      f.context = context; // записываем во вспомогательный класс
      req.setCharacterEncoding("UTF-8"); //уст. кодировку в запросах
      ServletContext app = getServletContext();
      appPath = app.getRealPath("/"); // получаем путь к папке проекта
      f.appPath = appPath; // записываем во вспомогательный класс
      session = req.getSession(); // получаем идентификатор сессии
      // макс. интервал бездействия 15 минут
      session.setMaxInactiveInterval(900);
      // получаем роль(статус) пользователя
      role = (String) session.getAttribute("role");
      // получаем идентификатор пользователя
      id = (String) session.getAttribute("id");
}
```

В основном модуле проверяется возможность начать тест (завершен ли предыдущий) и, если статус позволяет, меняем статус на «прохождение теста», создаем экземпляр класса информации о тесте и вопросах — **tt**,

сразу же заносим в него идентификатор теста и вызываем метод viewTestFrame(), чтобы показать заголок теста (название и кнопка «Начать тест»)

```
// основной модуль
protected void doModule(HttpServletRequest req, HttpServletResponse res)
                  throws ServletException, IOException, JDOMException{
      String status = (String) session.getAttribute("status");
      String login = (String) session.getAttribute("login");

      // если студент начал второй тест, не закончив первый
      if (role.equals(f.role_student) && (status.equals("busy.testing"))) {
            f.viewAlert(f.msg_test_unfinished, context, out);
            return;
      } else {
            status = "logged"; // учителю не важно
      }

      if (status == "logged") { // если статус позволяет
            current = -1; // текущий вопрос -1
            status = "busy.testing"; // статус "занят.тестирование"
            session.setAttribute("status", status); // сохранить в сессии
      } else { // иначе
            // получаем номер текущего вопроса из сессии
            current = ((Integer)
                        session.getAttribute("current")).intValue();
      }

      if (current == -1) { // если текущий вопрос -1
            // создаем экземпляр класса инф. о тесте и вопросе
            tt = new data();
            // заполняем идентификатор теста из запроса
            tt.testid = req.getParameter("testid");
            viewTestFrame(); // показать заголовок теста
      }
}
```

Помимо этого метод viewTestFrame() анализирует QTI XML файл теста (по идентификатору) и получает первичную информацию о тесте (количество вопросов, ссылки на файлы вопросов, название теста). Если файл теста не найден, то идентификатор теста удаляется из списка назначенных пользователю (файл appt.xml и вектор в сессии).

```
// показать заголовок теста
private void viewTestFrame() throws IOException, JDOMException {
// обработка ситуации, когда файл теста не найден (удален)
try {
      // загружаем qti xml файл теста
      Document doc = f.getXMLDocument(
                        appPath+"tests\\"+tt. testid+"\\test.xml");

      // получаем список ссылок на вопросы
      List itemRefList = f.getXMLItems("assessmentItemRef",
         f.getXMLItem("assessmentSection", f.getXMLItem("testPart", doc)));

      tt.questionsCount = itemRefList.size();
```

```
      for (int i = 0; i < tt.questionsCount; i++) {
            tt.itemsHref.add(i, ((Element)
                  itemRefList.get(i)).getAttribute("href").getValue());

            tt.itemsID.add(i, ((Element)
               itemRefList.get(i)).getAttribute("identifier").getValue());
            }
      tt.testTitle = f.getRootTitle(doc); // извлекаем заголовок теста
      // форма начала теста
      out.println("<br><table id='test_head' cellspacing='1'
                                                cellpadding='1'>");
      out.println("<form method='post' action='" + context+"/player'>");
      out.println("<tr id='calign'><td>"+tt.testTitle+"</td></tr>");
      out.println("<tr id='calign'><td><input type='submit'
                                               value='Начать тест'>");
      out.println("<input type='hidden' name='cmd'
                                                value='start'></td></tr>");
      out.println("</form>");
      out.println("</table>");

} catch (IOException e){ // если тест не найден
      f.viewAlert(f.msg_test_not_found, context, out);
      if (role.equals(f.role_tutor)){return;} // учителю не важно
            // удаляем из назначенных и из сессии
            delTestFromApptAndSession();
      }
      session.setAttribute("data", tt); // сохраняем в сессии
}
```

Важным в выводимой форме является скрытый (hidden) параметр cmd со значением «start», именно этот параметр будет обрабатываться, когда ученик начнет выполнение теста. Так как параметры передаются из формы методом POST, то для обработки этого запроса сервлет-контейнер вызовет метод doPost() нашего сервлета.

```
// реакция на HTTP POST запрос
protected  void  doPost(HttpServletRequest  req,  HttpServletResponse  res)
                                    throws ServletException, IOException{
            <…>
//обработка параметра cmd запроса
// start – начало теста

            String cmd = req.getParameter("cmd"); // команда
            if (cmd != null && cmd.equals("start")){
                  startTest(); // начать тест
            }

//обработка параметра giveAnswer (дан ответ) запроса

            String giveAnswer = req.getParameter("giveAnswer");
            if (giveAnswer == null) { // если пусто
                  resetData(); // сброс данных
                  parseAssessmentItem(); // парсинг очередного вопроса
                  render(req); // отрисовка вопроса
            } else { // иначе
                  processItem(req); // обработка вопроса
            }
            <…>
}
```

При получении параметра cmd со значением «start», метод doPost() вызывает метод startTest() — начать тест (начало самого процесса тестирования), основное назначение которого, добавление в список результатов (файл results.xml) новой записи, куда заносится идентификаторы результата и теста, название теста, время начала тестирования. Так же создается сам QTI XML файл отчета о прохождении тестирования (<идентификатор результата>.xml), куда записывается секция testResult

```
// начало теста
private void startTest() throws IOException, JDOMException {
      // извлекаем из сессии инф. о тесте и вопросах
      tt = (data) session.getAttribute("data");
// добавление информации в базу данных результатов пользователя
      Document doc = f.getXMLDocument(
                              appPath+"\\users\\"+id+"\\results.xml");
      // генерирование уникального идентификатора
      int last = Integer.valueOf(f.getItemValue(
                                          f.getXMLItem("last", doc)));
      last++;
      f.setItemValue(f.getXMLItem("last", doc), String.valueOf(last));
      resID = last; // идентификатор результата
// создание новой записи результата
      Element newResults = new Element("test");
      f.setItemAttr(newResults, "resid", String.valueOf(last));
      f.setItemAttr(newResults, "testid", tt.testFileName);
      f.setItemAttr(newResults, "title", tt.testTitle);
      // получение даты начала теста
      SimpleDateFormat sdf =  new SimpleDateFormat("yyyyMMdd'T'HHmmss");
      Calendar cal = Calendar.getInstance();
      String time = sdf.format(cal.getTime());
      f.setItemAttr(newResults, "date", time);
      f.addItem(newResults, doc); // добавление созданной записи
      // запись в базу
      f.dumpXMLDocument(doc, appPath+"\\users\\"+id+"\\results.xml");
      current = 0; // текущий вопрос 0
      session.setAttribute("current", current);  // сохраняем в сессии
// создаем начальный файл результата с полученным уникальным
идентификатором (в пределах пользователя)
      createResult(String.valueOf(last));
      docRes = f.getXMLDocument(
         appPath+"\\users\\"+id+"\\results\\"+String.valueOf(last)+".xml");
      nmRes = f.getNamespace(docRes); // получаем пространство имен
      // создаем секцию testResult
      testResult = new Element("testResult", nmRes);
      f.setItemAttr(testResult, "datestamp", time); // записываем дату
      f.setItemAttr(testResult, "normalMaximum",
               String.valueOf(tt.questionsCount)); // количество вопросов
      // сохраняем в сессии инф. о тесте и вопросах
      session.setAttribute("data", tt);
}
```

Сразу после проверки параметра cmd, проверяется параметр giveAnswer (дан ли ответ), если он пуст, то вызывается три метода resetDate() для сброса данных, parseAssessmentItem() для анализирования QTI XML файла первого вопроса в тесте, render() для отрисовки вопроса по полученным данным. Иначе, если параметр giveAnswer определен, то есть дан ответ на какой-то вопрос, то запускается обработка ответа processItem().

Метод parseAssessmentItem() помимо анализирования файла вопроса, добавляет в QTI XML файл результата очередную секцию itemResult (результат вопроса), куда заносит секцию responseVariable, в которую, соответственно, добавляются список идентификаторов правильных ответов correctResponse и список идентификаторов ответа ученика candidateResponse (пока пустой). Помимо responseVariable, в секцию itemResult добавляется клон секции itemBody из QTI XML файла вопроса.

```
// анализирование qti xml вопроса
private void parseAssessmentItem() throws ServletException, IOException,
                                          JDOMException{
//Загружаем очередной вопрос
      Document doc = f.getXMLDocument(appPath+"tests\\"+tt.testFileName+
                                "/" +tt.itemsHref.get(current).toString()
// Считываем responseDeclaration
      Element responseDeclaration = f.getXMLItem("responseDeclaration",
                                                                  doc);

// Считываем идентификаторы правильных ответов
      tt.answersID.removeAllElements();
      Element correctResponse = f.getXMLItem("correctResponse",
                                                    responseDeclaration);
      List correctList = f.getXMLItems("value", correctResponse);
      tt.answersCount = correctList.size();
      for (int i = 0; i < tt.answersCount; i++){
            tt.answersID.add(i, f.getItemValue((Element)
            correctList.get(i)));
      }

// Получение содержимого itemBody
      Element itemBody = f.getXMLItem("itemBody", doc);
      List cl = f.getXMLItems("", itemBody); // список всех подэлементов
      isTopFrame = true; // обработка верхнего заголовка
      int count = cl.size();
      for (i = 0; i < count; i++){
            // получение имени тэга
            tagName =  f.getItemName((Element) cl.get(i));
            // если элемент взаимодейтсвия choiceInteraction
            if (tagName.equals("choiceInteraction")) {
                  isTopFrame = false; // верхний заголовок кончился
                  // обработка choiceInteraction
                  parseInteraction((Element) cl.get(i));
```

```
            } else {
                  // обработка xhtml-контента
                  if (isTopFrame){
                        tt.topFrame += f.getXHTMLContent((Element)
                        cl.get(i));
                  } else {
                        tt.bottomFrame += f.getXHTMLContent((Element)
                        cl.get(i));
                  }
            }
      }

//====== Запись результатов: заполнение itemResult ==== begin ====

      // создаем новый элемент
      itemResult = new Element("itemResult", nmRes);
      // согласно qti стандарту заполняем
      Element responseVariable = new Element("responseVariable", nmRes);
      candidateResponse = new Element("candidateResponse", nmRes);
      f.addItem(candidateResponse, responseVariable);
      f.addItem((Element) correctResponse.clone(), responseVariable);
      f.addItem(responseVariable, itemResult);
      f.addItem((Element) itemBody.clone(), itemResult);
      f.setItemAttr(itemResult, "sequenceIndex",
                                                String.valueOf(current+1));
      f.addItem(itemResult, docRes);

//====== Запись результатов: заполнение itemResult ===== end =====

}
```

При анализировании QTI XML файла вопроса, как только анализатор дойдет до элемента с именем choiceInteraction (элемент взаимодействия — выборочный ответ), вызывается метод parseInteraction(), которому передается элемент choiceInteraction в качестве параметра на дальнейшую обработку. Основная задача метода parseInteraction() — получения информации о вопросе: сам вопрос, заголовки, варианты ответа, идентификаторы вариантов ответа, максимально возможное число выбранных ответов.

```
//обработка элемента взаимодействия
// e — элемент (имя тега — одного из элементов взаимодействия)
private void parseInteraction(Element e)throws IOException, JDOMException,
                                                DataConversionException {
      int xhtmlTagsCount;
      int j;

// Узнаем тип опроса (сколько вариантов ответа можно выбрать)
      Attribute at = e.getAttribute("maxChoices");
      if (at != null) {
            if (at.getIntValue() == 1) {
                  tt.Multiple = false;
                  tt.Single = true;
                  tt.maxChoices = 1;
            } else {
```

```
                tt.Multiple = true;
                tt.Single = false;
                tt.maxChoices = at.getIntValue();
            }
        } else {
            tt.Multiple = false;
            tt.Single = true;
            tt.maxChoices = 1;
        }

// Считываем вопрос к вариантам ответа
        Element promptElement = f.getXMLItem("prompt", e);
        tt.prompt = f.getVariableContent(promptElement);


        List scList = f.getXMLItems("simpleChoice", e);
        tt.choicesCount = scList.size(); // получение списка ответов

// Считываем идентификаторы вариантов ответов
        tt.choicesID.removeAllElements();
        for (j=0; j < tt.choicesCount; j++){
            tt.choicesID.add(j, f.getItemAttr((Element)
                                        scList.get(j),"identifier"));
        }

// Считываем варианты ответов
        tt.choices.removeAllElements();
        for (j=0; j < tt.choicesCount; j++){
            tt.choices.add(j, f.getVariableContent((Element)
                                                    scList.get(j)));
        }

}
```

После получения всех данных о вопросе вызывается метод render() — вывод оформленного вопроса на страницу.

```
// отрисовка вопроса
private void render(HttpServletRequest req){
        f.printHTMLTop(out);
        out.println("<table id='mt' align='center'><tr><td>");
        out.println("<center>");
        out.println("<table id='test_head' cellspacing='1'
                                                    cellpadding='1'>");
        // верхний заголовок
        out.println("<tr><td>"+tt.topFrame+"</td></tr>");
        out.println("<tr><td>"+tt.prompt+"</td></tr>"); // вопрос
        //форма для выбора ответа
        out.println("<form method='post' action='" + context+"/player'>");
        // в зависимости от указанного максимального числа ответов
        // radio – возможность выбора только одного варианта
        // checkbox – возможность выбора более одного варианта
        String card = (tt.Single == true)?"radio":"checkbox";
        for (i=0; i < tt.choicesCount; i++){
            out.println("<tr><td><input type='"+card+"' id='cbox'
                name='answer' value='"+tt.choicesID.get(i).toString()+"'>"
                +tt.choices.get(i).toString()+"</td></tr>");
        }
        out.println("<tr id='calign'><td>");
        out.println("<input type='submit' value='Ответить'>");
        out.println("<input type='hidden' name ='giveAnswer'
                                            value='"+current+"'>");
```

```
      out.println("</form>");
      out.println("</td></tr>");
      // нижний заголовок
      out.println("<tr><td>"+tt.bottomFrame+"</td></tr>");
      out.println("<tr id='calign'><td><font color='red'><b>" + tt.feedback
                              +  "</b></font></td></tr>"); // обратная связь
      out.println("</table>");
      out.println("</center>");
      out.println("</td></tr></table>");
      f.printHTMLBottom(out);
}
```

Стоит отметить, что именно в этой форме появляется скрытый параметр giveAnswer, который принимает значение номера текущего вопроса, и когда данные формы через метод POST попадут в метод doGet(), этот параметр будет определен и вызовется метод processItem() обработки ответа.

Основные задачи метода processItem(): фиксирование неадекватных действий (например, обновление окна с вопросом, возвращение к прошлым вопросам), обработка ответа с проверкой правильности количества выбранных ответов, проверка правильности самого ответа (и добавление балла к общему результату), запись идентификаторов выбранных учеником вариантов ответа в секцию candidateResponse документа результатов.

```
// обработка вопроса
private void processItem(HttpServletRequest req) throws IOException,
                                          JDOMException, ServletException {

      // извлекаем инф. о тесте и вопросах
      tt = (data) session.getAttribute("data");
      String giveAnswer = req.getParameter("giveAnswer"); // дан ответ
skip: { // блок перехода при ошибках

            // если ответ не от текущего вопроса
            if (!giveAnswer.equals(String.valueOf(current))) {
                  tt.feedback = "Зафиксировано обновление";
                  break skip;
            }

            // получение списка идентификаторов выбранных ответов
            String choicesResponse [] = req.getParameterValues("answer");
            if (choicesResponse == null) { // если пусто
                  tt.feedback = "Не выбран ответ!";
                  break skip;
            }

            //количество выбранных ответов
            int selectCount = choicesResponse.length;
```

```
            //если больше максимально разрешенного
            if ((selectCount > tt.maxChoices) && (tt.maxChoices > 0)) {
                  tt.feedback = "Слишком много ответов выбрано";
                  break skip;
            }

            // реализация проверки по типу match_correct
            int f, j;

            if (selectCount != tt.answersCount) {
                  tt.itemScore = 0;
            } else {
                  tt.itemScore = 1;
                  for (i = 0; i < tt.answersCount; i++){
                        f = 0;
                        for (j=0; j < selectCount; j++){
                              if (choicesResponse[j].equals(
                                          tt.answersID.get(i).toString())){

                                    f = 1;
                              }
                        }
                        tt.itemScore *= f;
                  }
            }

//====== Запись результатов: заполнение candidateResponse ==== begin ====

            // заполняем вектов выбранных ответов
            for (j=0; j<choicesResponse.length; j++){
                  candidateResponse.addContent(new
                                    Element("value",nmRes).addContent(
                                                choicesResponse[j]));
            }

            // получение даты и время ответа
            SimpleDateFormat sdf =  new
                                    SimpleDateFormat("yyyyMMdd'T'HHmmss");
            Calendar cal = Calendar.getInstance();
            String time = sdf.format(cal.getTime());
            itemResult.setAttribute("datestamp", time);

//====== Запись результатов: заполнение candidateResponse ===== end =====


            // увеличиваем на 1 текущий вопрос и сохраняем в сессии
            session.setAttribute("current", ++current);
            tt.currentQuestion = current;
            resetData(); // сброс данных
            // к общим баллам добавляем балл за вопрос
            tt.Score += tt.itemScore;
            if (current == tt.questionsCount) break skip; // если конец
            // анализируем очередной qti xml файл вопроса
            parseAssessmentItem();

} //skip

            if (current == tt.questionsCount) { // если текущий последний
                  // меняем статус, тест закончен
                  session.setAttribute("status", "logged");
                  // номер текущего вопроса -1
                  session.setAttribute("current", -1);
```

```
//====== Запись результатов: окончательная запись ==== begin ====

                  f.addItem(f.addItemValue(new Element("value", nmRes),
                                    String.valueOf(tt.Score)), testResult);
                  f.addItem(testResult, docRes);
                  f.dumpXMLDocument(docRes,
                                     appPath+"\\users\\"+id+"\\results\\" +
                                            String.valueOf(resID)+".xml");

//====== Запись результатов: окончательная запись ===== end =====

                  // если тест проходит студент
                  if (role.equals(f.role_student)){
                        // удалить тест из назначенных и из сессии
                        delTestFromApptAndSession();
                  }
                  // показать итоговый результат
                  f.viewSplash("Тест выполнен<br><br><b>Результат: </b>" +
                        tt.Score+ " / " + tt.questionsCount, context, out);


            } else { // иначе (если не последний вопрос)
                  render(req); // отрисовка вопроса
            }
}
```

В этом же методе проверяется был ли вопрос последним в тесте и, если вопрос последний, то выполняется окончательное формирование результата теста: добавляется итоговые результат в секцию testResult и полученный QTI документ записывается в XML файл. После этого выводятся итоговый результат теста, меняется статус ученика, так как тест завершен и он может проходить другие тесты, а идентификатор данного теста удаляется из списка назначенных (файл appt.xml и список в сессии).

При следующем входе в систему ученик может посмотреть полный отчет о прохождении теста.

Полный листинг данного сервлета, а так же листинги других сервлетов и JSP страниц прилагается на дискете, вместе со вспомогательными файлами системы (образ системы, готовой для установки, для этого необходимо скопировать папку test в директорию webapps сервлет-контейнера Apache Tomcat).

Рабочий вариант данной системы тестирования можно посмотреть по адресу http://217.77.52.113:8080/test/ (учитель: tutor, пароль: pass)

Заключение

Нельзя недооценить роль контроля и оценки знаний, а именно тестирования, в процессе обучения, а в рамках развития информационно-коммуникационных технологий особенно актуальна проблема автоматизации процесса тестирования — создание систем тестирования, которые позволяли бы на базе знаний в электронном виде, создавать тестовые задания и проводить тестирование участников обучения.

Для успешного развития отрасли образования в общем, и развития таких систем в частности, нельзя обосабливаться от остального мира, напротив, необходимо идти по пути интеграции с международной сферой образования. Поэтому использование *международных* стандартов в реализации автоматизированных систем тестирования считается нами необходимым условием.

Используя web-сервер Apache Tomcat, промышленный стандарт XML и язык программирования Java (с добавлением библиотеки JDOM) нами была создана пилотная система тестирования на основе международного стандарта взаимодействия и обмена данными в тестировании IMS QTI, поддерживающая создание и воспроизведение тестов с выборочными вариантами ответа. Среди возможностей системы: выполнение назначенных тестов, сохранения итогового результата и процесса выполнения теста, просмотр полного отчета о выполнении теста, создание новых тестов, редактирование имеющихся в системе тестов, поддержка групп пользователей.

Тестирование системы показало возможность использования ее как удобный и эффективный инструмент управления процессом обучения в условиях дистанционного обучения.

Таким образом, поставленные задачи дипломной работы решены, цель достигнута.

Литература

1. IMS Question & Test Interoperability Specification [Электронный ресурс] / Электрон. ст..— Режим доступа к ст.: http://imsglobal.org/question/index.html
2. Tarasov V.A. Using xml and the IMS QTI standard for the development of assessment tools / V.A. Tarasov, V.V. Tarasov, Kyurshunov A.S. // Mathematics and Science Education in the North-East of Europe: History, Traditions & Contemporary Issues Proceedings of the Sixth Inter-Karelian Conference Sortavala, Russia 11-14 September, 2003, 312 — 317 pp.
3. Тарасов В.А. Организация тестирования обучаемых в условиях локальной сети / Тарасов В.А., Кюршунов А.С. // Информатика и образование. 2003, №2, ст. 77—83
4. Тарасов В.А. Инструментальное программное средство для генерации тестов с выборочными ответами на языке JavaScript/ Тарасов В.А., Кюршунов А.С. // Информатика и образование. 2004, №7, ст. 113—115
5. Упрощение XML-программирования при помощи JDOM [Электронный ресурс]/ Электрон. ст..— Режим доступа к ст.: http://www.ibm.com/developerworks/ru/library/j-jdom/
6. Руководство по Web-сервисам. [Электронный ресурс]/ Электрон. ст..— Режим доступа к ст.: http://ru.sun.com/pdf/j2ee/WST.pdf
7. Тарасов В.А. Учебное пособие по спецкурсу Java / В.А. Тарасов, В.В. Тарасов, А.С. Кюршунов
8. Даконта, М. XML и Java 2. Библиотека программиста / М. Даконта, А. Саганич // Питер, 2001.
9. Язык XML Path (XPath) версия 1.0 [Электронный ресурс]/ Электрон. ст..— Режим доступа к ст.: http://www.citforum.ru/internet/xpath/index.shtml