Оглавление

[Введение 3](#_Toc326426906)

[Теоретическая часть 5](#_Toc326426907)

[Адаптивные обучающие системы 5](#_Toc326426908)

[Автоматизированные обучающие системы 5](#_Toc326426909)

[Компоненты интеллектуальной обучающей системы 8](#_Toc326426910)

[Адаптивный тестовый контроль 9](#_Toc326426911)

[Автоматное программирование 11](#_Toc326426912)

[Области применения автоматного подхода 12](#_Toc326426913)

[Основные понятия 14](#_Toc326426914)

[Автоматный подход и обучающие адаптивные системы 16](#_Toc326426915)

[Практическая часть 17](#_Toc326426916)

[Проект системы 17](#_Toc326426917)

[Структура адаптивной обучающей системы 17](#_Toc326426918)

[Описание процесса обучения 21](#_Toc326426919)

[Процесс контроля знаний 25](#_Toc326426920)

[Реализация 27](#_Toc326426921)

[Описание 27](#_Toc326426922)

[Автоматный подход 28](#_Toc326426923)

[Разработка модуля интеллектуальной проверки ответа 29](#_Toc326426924)

[Пользовательский интерфейс 30](#_Toc326426925)

[Заключение 32](#_Toc326426926)

[Список используемой литературы 33](#_Toc326426927)

# Введение

В настоящее время все больше и больше внимания уделяется разработке эффективных учебно-методических ресурсов. К таким ресурсам можно отнести различные электронные образовательные ресурсы, включающие в свой состав электронные учебники, системы контроля знаний и т.д., и интеллектуальные обучающие системы.

Подобные средства обучения позволяют реализовать концепцию удаленного обучения и повысить эффективность учебной деятельности за счет индивидуализации обучения. Казалось бы, реализовать связку «один ученик и много учителей» в современных условиях образования практически невозможно. Однако имеющийся опыт создания и использования компьютерных обучающих систем показал, что современные информационные технологии позволяют индивидуализировать процесс обучения и добиться высоких результатов [5]. Компьютерное обучение является альтернативой традиционным методам обучения, основанным на лекциях, практических и лабораторных занятиях. Различные исследования подтверждают факты эффективности компьютерного обучения, несмотря на то, что начальные затраты на организацию компьютерного обучения выше чем на традиционные технологии обучения.

Ряд коммерческих компаний разрабатывают высококачественные программные продукты учебного назначения [23]. Однако существует ряд причин, препятствующих широкому использованию подобного рода продуктов. Наиболее существенной причиной, на мой взгляд, является низкий уровень усвоения материала. Это связано с низкой адаптивностью обучающих систем, что является основополагающем фактором эффективности в процессе обучения.

В соответствии с классификацией, введённой Д. Харелом [25], любую программную систему можно отнести к одному из 3-х классов: трансформирующие, интерактивные и реактивные системы. Традиционными областями применения использования автоматов является – построение лексических, синтаксических анализаторов (класс трансформирующие системы) и задачи логического управления (класс реактивные системы)

В книге [15] выделено понятие «сложное поведение» для упрощения определения критерия применимости автоматного программирования. Иными словами, авторы советуют применять автоматный подход к объекту, если сущность в качестве реакции на входное действие может осуществить одно из нескольких выходных воздействий. Данное условие идеально подходит под обучение, так как оценка ученика производится только по одному критерию, а реакция системы зависит не только от результата приходящего в конкретный момент времени. *Поэтому наиболее актуальна разработка на основе автоматного подхода*.

*Целью* дипломной работы является разработка и реализация адаптивной обучающей системы позволяющей изучение любой предметной области.

*Задачами* данной работы являются:

* анализ основных принципов работы автоматизированных обучающих систем;
* разработка адаптивной модели обучения на основе автоматного подхода;
* разработка проекта адаптивной обучающей системы;
* реализация интеллектуальной системы проверки ответа;
* реализация системы.

# Теоретическая часть

# Адаптивные обучающие системы

Одной из задач обучающей системы, является обеспечение эффективности процесса обучения. В поисках интеллектуальных настроек и механизмов адаптации для системы, я решила обратиться к психологии обучения. Анализируя процесс обучения, возможности использования обучающих систем, нельзя не отметить важность индивидуального подхода к обучению.

Если рассмотреть классический процесс обучения, учитель, как человек интеллектуально развитый, без особого труда может найти нужную тактику обучения для каждого ученика, основываясь на его индивидуальном процессе усвоения материала, его умениях и знаниях. Такими же навыками должна обладать и обучающая система. Психология обучения исследует, прежде всего, процесс усвоения знаний и адекватных им умений и навыков. Ее задача состоит в выявлении природы этого процесса, его характеристик и качественно своеобразных этапов, условий и критериев успешного протекания.

Большинство исследований по психологии учения направлено на выявление закономерностей формирования и функционирования познавательной деятельности в условиях сложившейся системы обучения. В частности, накоплен богатый экспериментальный материал, вскрывающий типичные недостатки в усвоении различных научных понятий. Изучена также роль жизненного опыта учащихся, характера предъявляемого учебного материала в усвоении знаний[11].

С точки зрения психологии, важную роль в обучении играет память, т.е. такие важнейшие психические процессы, как запоминание и забывание, характеризующие усвоение знаний. В результате экспериментов психологов, были получены различные коэффициенты и зависимости, на основе которых были созданы первые модели обучения.

Идея автоматизации учебного процесса на данном этапе сводилась к использованию, главным образом, различных технических средств обучения, дополняющих учебный процесс. При этом технологичность  процесса обучения определялась объемом применения технических средств обучения как дополнительного средства обучения [2]. Постепенно исследователи переходили к идее применения технических средств обучения не как дополнения учебного процесса, а как устройства, берущего на себя некоторые функции учителя, т.к. технические средства обучения не обладали свойством управления учебным процессом, реализация с их помощью функций учителя, т.е. замена учителя техническим средством для управления или сопровождения хотя бы части учебного процесса было невозможно. В результате исследователи пришли к необходимости осмыслить сам учебный процесс, формализовать его и описать как технологический процесс.

Были исследованы различные способы его организации, основанные на различных педагогических методах. При этом основной принцип построения учебного процесса заключался в системе последовательных, четко описанных действий, выполнение которых ведет к заранее запланированной цели. Первым результатом этих исследований и одновременно основой последующих моделей обучения в начале 60-х годов XX века стала модель программируемого обучения, представленная во множестве изданий[11].

Сутью данной модели является адаптация учебного процесса под четко заданные цели. Цели представлены некоторым эталонным результатом, например, заданные правильные ответы. После сравнения результата с эталоном ставится оценка, которая является единственной характеристикой обучаемого. В зависимости от оценки выбирается следующий этап учебного процесса, при неудовлетворительной оценке могут быть выбраны и альтернативные способы изложения материала. Такие модели могут быть реализованы как линейными так и разветвленными схемами обучения.

## Автоматизированные обучающие системы

Стремительный рост объема и сложности изучаемого материала, необходимость его быстрого обновления при одновременном увеличении контингента обучаемых привели к тому, что по мнению многих ученых традиционная система обучения стала недостаточно эффективной [22], т.к. принципы, лежащие в ее основе, в условиях массового обучения не удается реализовать.

Идея индивидуального обучения и повышения его качества существовала задолго до появления компьютеров. В 1927 г. американский ученый С.Пресси использовал автоматические устройства для проверки правильности ответов учащихся на тестовые вопросы. Работая независимо и в совершенно различных областях, проф. Скиннер и доктор Н. Краудер заложили основы программированного обучения [2].

Программированное обучение может быть сведено к 4-м основным принципам [2].

1. Сначала определяются учебный материал, процесс или навыки, которым собираются обучить, затем они анализируются и разлагаются на составные элементы.

2. После этого материал предоставляется обучаемому шаг за шагом в тщательно продуманной последовательности.

3. На каждом шаге ученику дается ровно столько информации, сколько нужно, чтобы обеспечить его активную реакцию (в виде какого-то действия, которое показывает, что он следует данным ему указаниям), прежде, чем пойдет дальше.

4. Учащийся получает немедленное подтверждение правильности своего ответа, работает в своем собственном темпе и сам контролирует успешность своего обучения.

Компьютерные технологии позволяют использовать уникальные методы и средства для улучшения процесса обучения, организации новых форм передачи и контроля знаний, умений и навыков.

Качественно новым этапом в создании средств индивидуального и реально-игрового обучения явились современные автоматизированные обучающие системы (АОС), реализованные на ЭВМ с аппаратурой и программным обеспечением диалогового взаимодействия. Автоматизированная обучающая система – это компьютерное педагогическое программное средство, предназначенное для предъявления новой информации, усвоения навыков и умений, промежуточного и итогового тестирования, располагающее надежной системой помощи, как по самой обучающей программе, так и по изучаемому предмету [2]. Подобные системы обладают возможностью некоторой подстройки к обучаемому (его уровню знаний, скорости и пути продвижения по изучаемому материалу и т.д.), развитой системой сбора и обработки статистической информации о каждом обучаемом, группе и потоке обучаемых (в том числе накапливает информацию о часто встречающихся ошибках при работе с обучающей системой и ошибках обучаемых по изучаемой теме или дисциплине).

Задача автоматизации – повышение эффективности обучения, складывающееся из нескольких составляющих: более прочное усвоение материала, больший объем знаний, меньшее время на их усвоение.

Повышение эффективности обучения при использовании обучающих систем подтверждено рядом исследований [13]. Этому способствуют следующие факторы.

1. Индивидуализация обучения. Наиболее эффективным и наименее экономным является индивидуальное обучение (один преподаватель и один ученик), а самой экономичной, но и наименее эффективной системой - массовое обучение. Внедрение обучающих систем позволяет совместить достоинства индивидуального обучения (в смысле эффективности) и массового (в смысле экономичности).
2. Интенсификация обучения. Она достигается за счет индивидуальности обучения (толпа всегда идет медленнее одного человека), а также за счет того, что обучаемый не привязан ко времени занятия и к преподавателю, а может заниматься в удобное для себя время.
3. Использование выразительных средств вычислительной техники. К ним относятся наглядность, наличие средств моделирования объектов и процессов и т.п.
4. Возможность организации постоянного контроля степени усвоения знаний, способствующего более прочному закреплению материала.

## Компоненты интеллектуальной обучающей системы

Специфическими компонентами ИОС, отображающими ситуацию обучения, выступают [15]:

а) *модель обучаемого;*

б) *модель процесса обучения* (набор стратегий обучения);

в) *интерфейс* между экспертной подсистемой и прочими блоками ИОС.

Компонент «модель обучаемого» должен содержать информацию об индивидуальных особенностях студента, предпочитаемых им стратегиях обучения, типичных ошибках. Здесь необходимо организовать представление текущего уровня обученности, диагностику текущих знаний студента. Следует указать информацию о том, чего он не понимает, а также сведения о предпочтительной стратегии обучения (обучение на примерах, обучение по аналогии и т.п.).

Компонент «модель процесса обучения» обеспечивает предъявление информации и оценку качества деятельности студента. Здесь присутствуют знания о планировании и организации процесса обучения, об общих и частных методиках обучения, обеспечивается реализация различных режимов обучения:

а) тренировка обучаемого, например, в процессе развивающей игры, когда за счет изменения условий игры у студента формируются требуемые навыки и умения;

б) постановка тестовых задач, по результатам решения которых можно судить об уровне подготовки и ошибках обучаемого;

в) вопросно-ответные процедуры, в ходе которых обучаемого побуждают к формированию цепочек рассуждений, причем могут использоваться такие модели, которые позволяют обучаемому самому открывать некоторые правила или оценивать факты.

## Адаптивный тестовый контроль

Адаптивный тестовый контроль и адаптивное обучение представляют собой современный компьютерный вариант реализации известного педагогического принципа индивидуализации обучения – одного из самых важных для подготовки квалифицированных специалистов. Данный принцип не мог быть реализован сколько-нибудь эффективным образом в условиях традиционных классно-урочных форм занятий – лекционных и групповых. В современном варианте адаптивные формы появились недавно, как результат взаимосвязи трех прогрессивных тенденций: компьютеризации образования, возрастания роли аутодидактики (самообучение, самообразование) и развития теории педагогических измерений [1].

Важность адаптивного контроля вытекает из соображений рационализации традиционного тестирования. Подготовленному учащемуся нет необходимости дaвaть легкие задaния, потому что высока вероятность их прaвильного решения. Лёгкие материaлы не обладают заметным развивающим потенциалом, в то время как трудные задания у большинства учащихся снижают учебную мотивацию. Аналогично нет смысла давать трудные задания слабому ученику из-за высокой вероятности неправильного решения. Использование заданий, соответствующих уровню подготовленности, существенно повышает точность измерений и минимизирует время индивидуального тестирования до 5 – 10 минут [12].

Таким образом, адаптивный тест представляет собой вариант автоматизированной системы тестирования, в которой заранее известны параметрами трудности и дифференцирующей способности каждого задания. Эта система может быть создана в виде компьютерного банка заданий, упорядоченных в соответствии с интересующими характеристиками заданий. Самая главная характеристика заданий адаптивного теста – это уровень их трудности, полученный опытным путем, что означает: прежде чем попасть в банк, каждое задание проходит эмпирическую апробацию на достаточно большом числе типичных учащихся интересующего контингента.

В [26] выделяются три варианта адаптивного тестирования. Первый называется пирамидальным тестированием. При отсутствии предварительных оценок всем испытуемым дается задание средней трудности и уже затем, в зависимости от ответа, каждому испытуемому дается задание легче или труднее. На каждом шаге можно использовать правило деления шкалы трудности пополам. Второй вариант – начало контроля с любого подходящего уровня трудности, с постепенным приближением к реальному уровню знаний. Третий вариант – тестирование проводится посредством банка заданий, разделенных по уровням трудности. При правильном ответе следующее задание берется из верхнего уровня, при неправильном ответе – из нижнего.

Адаптивное обучение позволит учить материал без привычных для массового образования многочисленных пробелов в индивидуальной подготовленности учащихся и студентов, достигать требуемой структуры и желаемого уровня знаний.

# Автоматное программирование

Автоматное программирование, иначе называемое «программирование от состояний» или «программирование с явным выделением состояний» – это метод разработки программного обеспечения, основанный на расширенной модели конечных автоматов и ориентированный на создание широкого класса приложений.

Программирование с использованием автоматов имеет достаточно богатую историю развития. Различные аспекты и понятия, связанные с этой идеей, рассматривались в работах многих авторов с самых разных точек зрения и применительно к различным конкретным вопросам. Программирование от состояний рассматривается как один из основных стилей программирования в книге Непейвода Н. Н. «Стили и методы программирования».

Как целостная парадигма разработки ПО, автоматное программирование сформировалось, в основном, благодаря усилиям одного из авторов книги [15], который в 1991 г. предложил технологию для поддержки этого стиля программирования. Однако полное и исчерпывающее изложение сути автоматного программирования как парадигмы и метода разработки программных систем в целом в настоящий момент отсутствует. Эта книга может считаться первым шагом к восполнению этого пробела.

Термин «автоматное программирование» родился в 1997 г. на конференции по мультиагентным системам, проходившей в поселке Ольгино под Санкт-Петербургом, и был впервые использован во введении к работе Шалыто А. А. «SWITCH-технология». На английский язык этот термин переводится как automata-based programming. [15]

## Области применения автоматного подхода

Конечные автоматы в программировании традиционно применяются при создании компиляторов, которые относятся к классу трансформирующих систем. Автомат здесь понимается как некое вычислительное устройство, имеющее входную и выходную ленты. Перед началом работы на входной ленте записана строка, которую автомат далее посимвольно считывает и обрабатывает. В результате обработки автомат последовательно записывает некоторые символы на выходную ленту [21].

Другая традиционная область использования автоматов – задачи логического управления [21] – является подклассом реактивных систем. Здесь автомат – это, на первый взгляд, совсем другое устройство. У него несколько параллельных входов (чаще всего двоичных), на которые в режиме реального времени поступают сигналы от окружающей среды. Обрабатывая эти сигналы, автомат формирует значения нескольких параллельных выходов.

Таким образом, даже традиционные области применения конечных автоматов охватывают принципиально различные классы программных систем. Круг задач, при решении которых целесообразно использовать автоматный подход, значительно шире и включает создание программных систем, принадлежащих всем трем перечисленным классам. Однако, автоматные модели, используемые при создании различных видов программных систем, могут отличаться друг от друга.

Критерий применимости автоматного подхода лучше всего выражается через понятие «сложное поведение». Неформально можно сказать, что сущность (объект, подсистема) обладает сложным поведением, если в качестве реакции на некоторое входное воздействие она может осуществить одно из нескольких выходных воздействий. При этом существенно, что выбор конкретного выходного воздействия может зависеть не только от входного воздействия, но и от предыстории. Для сущностей с простым поведением реакция на любое входное воздействие зависит только от этого воздействия.

Одна из центральных идей автоматного программирования состоит в отделении описания логики поведения (при каких условиях необходимо выполнить те или иные действия) от описания его семантики (собственно смысла каждого из действий). Кроме того, описание логики при автоматном подходе жестко структурировано. Эти свойства делают автоматное описание сложного поведения наглядным и ясным.

По мнению авторов [15], основная рекомендация по применению автоматного программирования: использовать автоматный подход при создании любой программной системы, в которой есть сущности со сложным поведением. Отметим, что таким свойством обладает практически любая серьезная система. Однако обычно не все компоненты системы характеризуются сложным поведением. Поэтому целесообразно использовать автоматный подход при создании только тех компонентов системы, которые являются сущностями со сложным поведением.

Также отметим некоторые проекты, реализованные в рамках автоматного подхода [15]:

* автоматная реализация интерактивных сценариев образовательной анимации с использованием MacromediaFlash;
* построение визуализаторов алгоритмов для обучения дискретной математике и программированию;
* совместное использование теории построения компиляторов и Switch-технологии;
* управление различными технологическими процессами и объектами
* примеры клиент-серверных приложений;
* построение пользовательских интерфейсов;
* управление в задачах логистики.

## Основные понятия

Базовым понятием автоматного программирования является «состояние». Это понятие в том смысле, как оно используется в описываемой парадигме, было введено А. Тьюрингом и с успехом применяется во многих развитых областях науки, например, в теории управления и теории формальных языков [15].

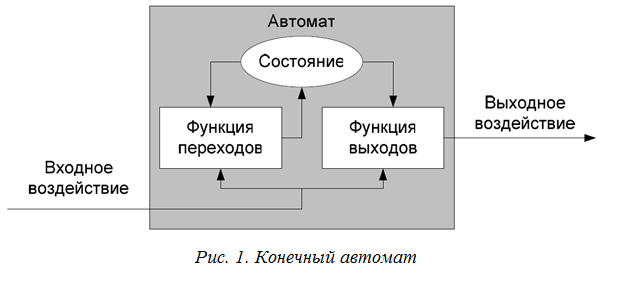
Основное свойство состояния системы в момент времени *t0* заключается в «отделении» будущего *(t >t0)* от прошлого *(t <t0)* в том смысле, что текущее состояние несет в себе всю информацию о прошлом системы, необходимую для определения ее реакции на любое входное воздействие, формируемое в момент времени *t0*.

При описании понятия «сложное поведение» упоминалось, что реакция сущности со сложным поведением на входное воздействие может зависеть, в том числе, и от предыстории. При использовании понятия «состояние» знание предыстории более не требуется. Состояние можно рассматривать как особую характеристику, которая в неявной форме объединяет все входные воздействия прошлого, влияющие на реакцию сущности в настоящий момент времени. Реакция зависит теперь только от входного воздействия и текущего состояния.

В [15] отмечается, что эффективность объектно-ориентированного подхода к разработке программного обеспечения объясняется тем, что для человека естественно мыслить в терминах объектов (сущностей) и взаимодействия между ними. У автоматного подхода, также есть все шансы стать эффективным, так как люди живут в состояниях (например, спят или бодрствуют, сыты или голодны), и в зависимости от текущего состояния по-разному реагируют на внешние раздражители.

Понятие входное воздействие также является одним из базовых для автоматного программирования [15]. Чаще всего, входное воздействие – это вектор. Его компоненты подразделяются на события и входные переменные в зависимости от смысла и механизма формирования. Совокупность конечного множества состояний и конечного множества входных воздействий образует (конечный) автомат без выходов. Такой автомат реагирует на входные воздействия, определенным образом изменяя текущее состояние. Правила, по которым происходит смена состояний, называют функцией переходов автомата.

То, что в автоматном программировании собственно и называется (конечным) автомат (рис. 1), получается, если соединить понятие автомата без выходов с понятием «выходное воздействие». Такой автомат реагирует на входное воздействие не только сменой состояния, но и формированием определенных значений на выходах. Правила формирования выходных воздействий называют функцией выходов автомата.



Конечный автомат состоит из конечного управления и входной ленты, разбитой на ячейки. В каждой ячейке записан один символ из входного алфавита Σ, и все они образуют конечную входную цепочку. Конечное управление первоначально находится в состоянии q0 и сканирует крайнюю левую ячейку ленты. По мере чтения входной цепочки слева направо автомат переходит в другие состояния из множества Q. Если, прочитав входную цепочку, автомат оказывается в некотором конечном состоянии из множества F, то говорят, что он принял ее. Множество цепочек, принимаемых конечным автоматом, называется языком, распознаваемым данным конечным автоматом.

Конечные автоматы не могут распознавать все языки, порождаемые грамматиками, но языки, распознаваемые ими, являются в точности языками, порождаемыми грамматиками типа 3. В последующих главах мы познакомимся с распознавателями для языков типа 0, 1 и 2.

Конечным автоматом называется формальная система

*M = (Q, Σ, δ, q0, F)*

, где:

*Q* — конечное непустое множество состояний;

Σ — конечный входной алфавит;

*δ* — отображение типа Q × Σ → Q;

*q0∈ Q* — начальное состояние;

*F ⊆ Q* — множество конечных состояний.

Запись *δ(q, a) = p*, где *q*, *p ∈ Q* и *a∈ Σ*, означает, что конечный автомат *M* в состоянии *q*, сканируя входной символ *a*, продвигает свою входную головку на одну ячейку вправо и переходит в состояние *p*.

Область определения отображения *δ* можно расширить до *Q* × Σ\*  следующим образом: *δ’(q, ε) = q, δ’(q, xa) = δ(δ’(q, x), a)* для любого *x∈* Σ\* и *a∈* Σ.

Таким образом, запись *δ’(q, x) = p* означает, что конечный автомат *M*, начиная в состоянии *q∈ Q* чтение цепочки *x∈* Σ\*, записанной на входной ленте, оказывается в состоянии *p∈ Q*, когда его входная головка продвинется правее цепочки *x*. Определенная таким образом модель конечного автомата называется детерминированной.

Цепочка *x ∈ Σ*\* принимается конечным автоматом *M*, если *δ(q0, x) = p* для некоторого *p ∈ F*. Множество всех цепочек *x ∈* Σ\*, принимаемых конечным автоматом M, называется языком, распознаваемым конечным автоматом M, и обозначается как *T(M),* т. е.

*T(M) = {x ∈ Σ\* | δ(q0, x) = p* при некотором *p ∈ F}.*

Диаграмма состояний конечного автомата состоит из узлов, представляющих состояния, и из ориентированных дуг, определяющих возможные переходы, которые зависят от входных символов. Так, если *δ(q,a)= p*, то из узла, представляющего состояние *q*, в узел, представляющий состояние *p*, проводится дуга, помеченная входным символом *a*. На рис.5 дана диаграмма состояний конечного автомата. Двойным кружком выделены конечные состояние, а кружком с треугольником – начальное.

## Автоматный подход и обучающие адаптивные системы

В настоящее время экспертными системами наиболее широко применяемого типа являются системы, основанные на правилах. В системах, основанных на правилах, знания представлены не с помощью относительно декларативного, статического способа (как ряд истинных утверждений), а в форме многочисленных правил, которые указывают, какие заключения должны быть сделаны или не сделаны в различных ситуациях. Система, основанная на правилах, состоит из правил IF–THEN, фактов и интерпретатора, который управляет тем, какое правило должно быть вызвано в зависимости от наличия фактов в рабочей памяти.

У продукционной модели есть существенные недостатки:

* при накоплении достаточно большого числа (порядка нескольких сотен) продукций они начинают противоречить друг другу.
* отсутствие внутренней структуры и зависимости шагов дедуктивного вывода от стратегии вывода, что делает ее трудно интерпретируемой.

Исходя из недостатков продукционного, и преимуществ автоматного подходов было принято решение создать экспертную систему, использующую автоматы.

# Практическая часть

## Проект системы

### Структура адаптивной обучающей системы

Обучающую систему можно разделить на следующие основные компоненты (рис 2):

* Модуль подготовки учебного материала
* Модуль обучения
* Модуль тестирования
* Модуль работы с учетными записями учеников
* Пользовательский интерфейс

1. *Модуль подготовки учебного материала* позволяет преподавателю подготовить систему для изучения конкретного курса. С его помощью подготавливается материал для обучения, тестирования, а также устанавливаются глобальные параметры курса, этот материал необходим для корректной работы системы и используется остальными компонентами в процессе обучения.

Материалом для обучения являются лекции представленные в html файлах, разделённые на главы (разные html файлы), файлы содержащие в себе различные файлы должны быть единообразно названы и отличаться номерами глав и уровнями сложности. Например для третей главы изучения курса Oracle «Выражения в Oracle SQL» с простым изложением материала (уровень сложности = 0) название файла может быть следующим – «Oracle\_3\_0.htm» или «Oracle\_уровень0\_тема3.html», ключевым моментом является указание номера темы и уровня в названии файла. Для подготовки лекционных материалов необходимо, что бы все лекции находились в одном каталоге, а затем указать данный каталог при работе с модулем.

Материал для тестирования – это наборы вопросов с ответами для различных тем и уровней сложности. Материалы заполняются посредством диалога с помощью консольного интерфейса.

Таким же способом, как и материал для тестирования, заполняются глобальные параметры курса: название курса, количество глав, максимальный и минимальный уровень сложности, количество вопросов в группе, информация о преподавателе, нормальное время прохождения теста.

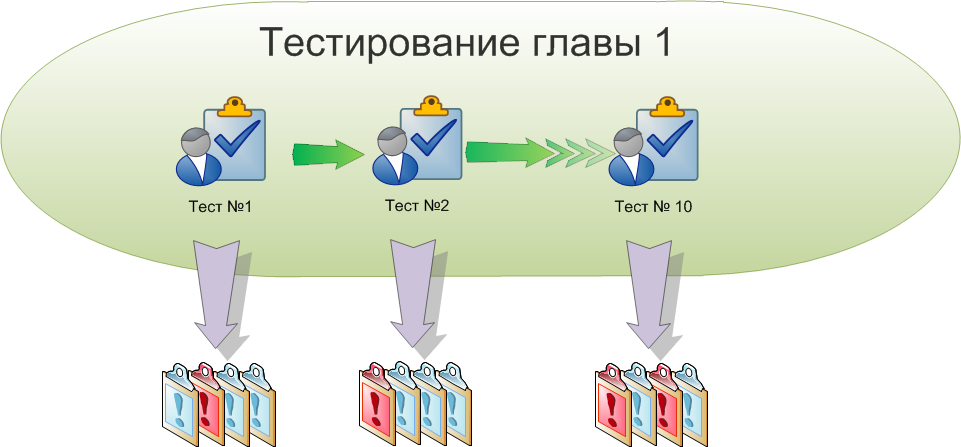
*Рис.2. Структура адаптивной обучающей системы*

1. *Модуль обучения* − управляет процессом обучения и фактически взаимодействует с другими модулями для решения задач возникающих на этапе обучения.

Основные задачи:

* подбор и выдача материала для обучения в соответствии с местом ученика в учебном процессе;
* инициализация процесса тестирования с предоставлением необходимых данных об ученике, передача управления модулю тестирования;
* формирование записи ученика на основе данных пользователя системы;
* принятие решения о сохранении записи ученика, в случае сбоя системы, а также в конце сеанса работы, передача модулю работы с учетными записями учеников.

1. *Модуль тестирования* обеспечивает контроль знаний обучаемого. Организует процесс адаптивного тестирования в соответствии с местом ученика в учебном процессе. Схему тестирования одной главы можно посмотреть на рисунке, представленном ниже (рис 3).



*Рис.3. Схема тестирования одной главы*

1. *Модуль работы с учетными записями* пользователей предоставляет следующие возможности:

* обеспечивает сохранность данных ученика между сессиями обучения: сохранение всей истории обучения, сохранение изученных тем и результатов тестирования;
* возможность загружать, сохранять и создавать новых пользователей;
* обеспечивает модулю обучения доступ к истории обучения пользователя.

1. *Пользовательский интерфейс* предназначен для повышения удобства пользования системой, за счет использования привычного оконного интерфейса.

Основной задачей данного модуля является обеспечение взаимодействия обучающей системы с пользователем (предоставление учебного материала, организация процесса тестирования). То есть пользовательский интерфейс позволяет в диалоговом режиме: просмотреть ошибки при ответе, изучить материал, просмотреть задание и ввести ответ, при необходимости приостановить обучение, предоставить пользователю выбор о создании нового пользователя или использовании существующей учетной записи.

### Описание процесса обучения

Схема обучения следующая: сначала студент знакомится с материалом урока, после чего проходит тестирование по изученной теме. На всем этапе обучения система, словно учитель, внимательно следит за учеником, а в конце занятия подводит итоги.

*Рис.4. Итерация обучения*

Весь процесс обучения контролируются модулем обучения. Именно этот модуль собирает необходимые данные на всех этапах обучения, на основе которых формируется запись конкретного ученика. Таким образом, запись ученика отражает его состояние в ходе обучения и включает в себя следующие характеристики:

* имя пользователя;
* название изучаемого курса;
* номера и уровни пройденных глав;
* результаты прохождения тестов;
* времена прохождения тестов.

Обучение происходит итеративно, пока обучаемый не изучит последнюю тему, на каждой итерации пользователю предлагается изучить лекционный материал и пройти по данному материалу тест. После окончания теста система принимает решение об изменении темы и уровня предлагаемого ученику материала, то есть система может перевести обучаемого:

* на следующий урок без изменения уровня сложности;
* прохождение урока заново без изменения сложности;
* на следующий урок с повышением уровня сложности;
* на повторение только что пройденного материала с понижением сложности;
* на следующий урок с понижением уровня сложности.

Решение о дальнейшем ходе обучения принимается с помощью автомата на основе критериев пользователя:

* средний результат выполнения тестов;
* скорость выполнения теста;
* результат последнего выполненного теста.

Конечное множество состояний автомата (Q) состоит из 5 состояний. Состояние определяется парой целых чисел, каждое из которых может принимать значение –1, 0, 1. Первое число означает изменение уровня сложности, а второе − изменение темы. То есть состояние «0 1» означает перевод обучаемого на следующий урок без изменения уровня сложности, состояние «–1 0» − повторение только что пройденного материала с понижением сложности и т.д.

Стартовым состоянием является q0 = {0 1}.

Финальными вершинами являются F = {0 1}, {1 1}, {–1 1}.

Допустимый входной алфавит (∑) состоит их троек (X,Y,Z), где

* X – значение, характеризующее скорость выполнения теста. Возможные значения:
* 0 – обучаемый выполнил тест медленнее установленного норматива
* 1 – обучаемый выполнил тест быстрее установленного норматива;
* Y – значение, отражающее средний результат выполнения тестов. Возможные значения:
* 3 – средний результат 90 – 100%
* 2 – средний результат 60 – 90%
* 1 – средний результат 0 – 60%
* Z – результат выполнения последнего теста. Возможные значения:
* 3 – результат 90 – 100%
* 2 – результат 60 – 90%
* 1 – результат 0 – 60%

Функция переходов автомата (δ) – представлена в таблице ниже. Символ «А» означает любое из допустимых значений, а символ косой черты означает логическое «или».

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 1 | 0 0 | 1 1 | –1 0 | –1 1 |
| 0 1 | 1 2 2  1 2 3  0 2 2  0 2 3  0 3 2  0 3 3 | A A 1 | 1 3 2  1 3 3 |  | 0 1 2  0 1 3  1 1 2  1 1 3 |
| 0 0 | AA3  1 2 2  1 3 2  0 3 2 |  |  | AA1  0 2 2  1 1 2  0 1 2 |  |
| 1 1 | AA2/3 | A A 1 |  |  |  |
| –1 0 | AA3 |  |  | A A 1/2 |  |
| –1 1 | AA2/3 | A A 1 |  |  |  |

*Таблица 1. Табличное представление автомата для определения состояния ученика в учебном процессе*

На основе состояния автомата определяет тему, предлагаемую пользователю для изучения, и уровень сложности учебного материала.



*Рис. 5. Графическое представление автомата для определения состояния ученика в учебном процессе*

### Процесс контроля знаний

Задания по каждой изучаемой теме делятся на несколько групп (высокой, средней, низкой сложности и т.п.), количество которых заранее определено преподавателем. Для каждого теста существует определённое количество групп вопросов (считаем, что одна такая группа соответствует неделимой части в изучаемой теме, назовём это разделом, то есть N вопросов в одном разделе). В рамках одного раздела определённой темы и уровня сложности, содержатся вопросы, направленные на выявление знаний ученика по данному разделу.

На этапе тестирования ученик должен будет пройти проверку по всем разделам пройденной темы. При очередной проверке изучения раздела, ученику предлагается ответить на вопрос. Если ученик отвечает неправильно на первый вопрос, то ему будет предлагаться вопросы пока он (ученик) не ответит правильно, либо не кончатся вопросы, вопросы выбираются случайным образом.

Для определения знаний ученика по одной теме – находится среднее значение результатов ответа ученика по главам из данного раздела. А результат теста по разделу вычисляется по формуле (если поступил правильный ответ)

*(100 + 30 \* (i* – *1) ) / i*

, где i – количество предлагаемых вопросов. Если от ученика не поступил правильный ответ и вопросы в разделе закончились – ученику присваивается результат 0.

Адаптивность контроля знаний выражается в том, что вариант очередного задания, предлагаемого ученику, зависит от места ученика в процессе обучения и от знаний ученика, показанных в ходе сеанса тестирования. Модуль тестирования, так же как и модуль обучения является конфигурируемым, то есть эксперт предоставляет тестовые задания различных уровней сложности, на основе которых происходит процесс тестирования.

Необходимо отметить, что в модуль тестирования включен немаловажный компонент – интеллектуальной проверки ответа. Под интеллектуальностью, подразумевается способность системы указывать на ошибку пользователя в веденном им ответе. Исходя из спецификации предметной области, задача «проверки ответов» сводится к задаче «проверка формул».

## Реализация

### Описание

Проект адаптивной обучающей системы реализован на высокоуровневом языке программирования Java.

Java  − объектно-ориентированный язык программирования, разработанный компанией [Sun Microsystems](http://ru.wikipedia.org/wiki/Sun_Microsystems) (в последующем, приобретённой компанией Oracle). Приложения Java обычно [компилируются](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) в специальный [байт-код](http://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), поэтому они могут работать на любой [виртуальной Java-машине](http://ru.wikipedia.org/wiki/Java_Virtual_Machine) (JVM) независимо от [компьютерной архитектуры](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0). Еще одной причиной для выбора именно языка java является возможность быстрого создания web-интерфейса на основе созданного приложения.

Важной особенностью системы является возможность использовать систему для изучения различных предметных областей. Для того что бы использовать систему для обучения какого либо предмета, необходимо инженеру по знаниям (преподаватель) подготовить обучающий материал, тестовый материал и критерии оценки обучаемого. Для проверки работоспособности системы предметом обучения была использован учебный материал «Математика», а именно «Производные».

### Автоматный подход

Существуют различные способы реализации конечного автомата – с явно заданной таблицей переходов, в автоматном стиле, на основе объектно-ориентированной парадигмы. До момента реализации системы, самым удачным я считал способ, при котором используется оператор switch, осуществляющий выбор по текущему номеру состояния. При всей своей простоте, реализация оператором switch совсем не вписывается в объектно-ориентированную парадигму и из-за этого не позволяет в полной мере использовать возможности современных языков, поэтому был выбран именно объектный подход.

Особенности использования объектного подхода:

* автомат задается также наглядно и лаконично, как и при использовании оператора switch;
* объектно-ориентированная реализация;
* поддерживает все возможности автоматного программирования.

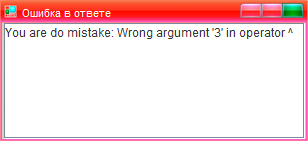
### Разработка модуля интеллектуальной проверки ответа

Для проверки ответа ученика разработан алгоритм сопоставления двух формул (s1 – ответ ученика и s2 – эталон):

* удаление пробелов из обеих формул;
* перевод формул в инфиксную запись;
* построение деревьев;
* сравнение деревьев полным обходом дерева (preorder traversal [27]) с допуском ассоциативности операций.

Если пользователем было введено некорректное выражение, то система выдаст соответствующее сообщение, о не корректности того или иного оператора или же операнда введённого пользователем, без какой либо подсказки об эталонной формуле.

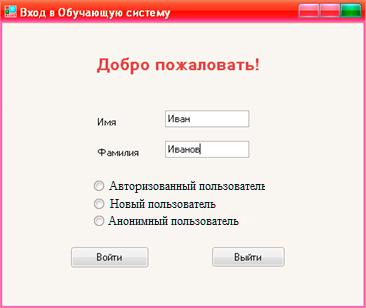
Для наглядности посмотрим работу данного модуля на примере: допустим, что правильным ответом является выражение «2\*x+y^2», а пользователь ввёл выражение «y^3+2\*x», тогда система выдаст сообщение о не корректности аргумента у оператора «^» (рис. 6). Нужно отметить, что анализатор принимает во внимание ассоциативность операций.

**

*Рис. 6. Окно ошибки в процессе обучения*

### Пользовательский интерфейс

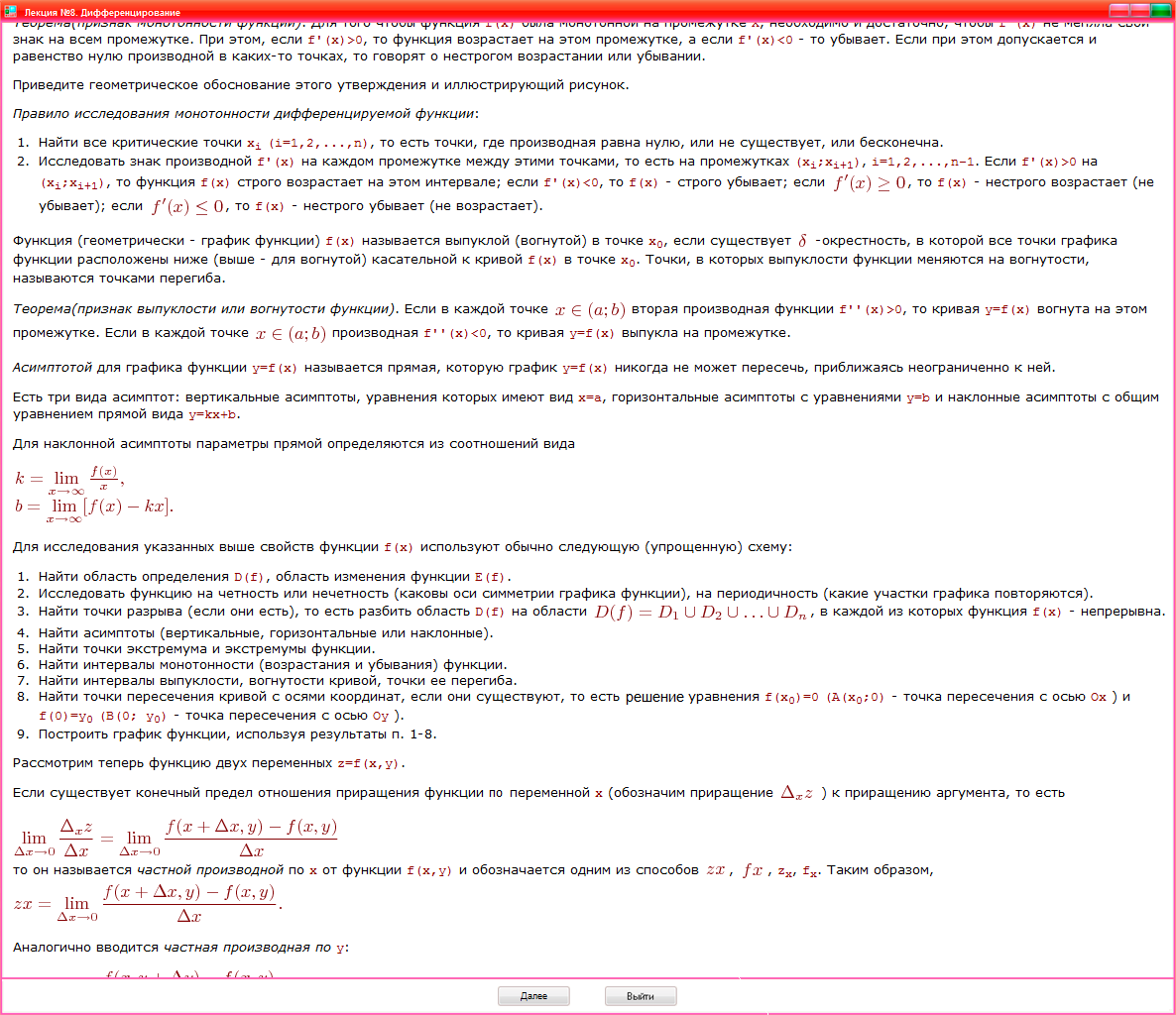
Пользователь, впервые начиная работу с программой, должен авторизоваться в системе. Тем самым он создает свою учетную запись, хранящую всю информацию о студенте. Изначально студенту присваивается стратегия средней сложности обучения.

****

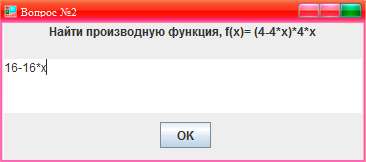
*Рис.7. Вход в обучающую системую*

Процесс обучение может продолжаться в течении нескольких сеансов работы в обучающей системе.

В режиме обучения пользователю предоставляется материал в оконном режиме содержащий обучающий материал (текст, графика) с HTML разметкой (рис 8). Тестирование производиться последовательной выдачей вопросов (рис 9).



*Рис.8. Активное окно системы во время изучения лекционного материала*



*Рис.9. Активное окно системы во время тестирования*

# Заключение

По итогам проделанной работы была достигнута главная цель – разработана конфигурируемая адаптивная обучающая система с применением автоматного программирования, гибко приспосабливающаяся под уровень подготовленности ученика.

На всем этапе работы АОС собирает данные об ученике, на основе которых получает количественные значения характеристик обучаемого.

С помощью полученных характеристик система выстраивает модель ученика. В системе нет заранее сформированных сценариев обучения. Есть только модель обучения, в рамках которой производиться обучение. Такое построение последовательности курса обучения обеспечивает обучаемого индивидуально спланированной последовательностью уроков.

Адаптивным является не только процесс обучения, но и процесс контроля знаний. Адаптивность выражается в изменении относительных пропорций в предъявлении легких и трудных заданий

Благодаря индивидуальному, а не массовому подходу эффективность обучения при подобной самостоятельной работе повышается.

# Список используемой литературы

1. Аванесов В.С. Адаптивное обучение и адаптивный тестовый контроль // Теория и методика педагогических измерений, 2003. http://testolog.narod.ru/Theory41.html
2. Акманов Н.И. Автоматизированные обучающие системы как средство эффективной подготовки специалистов // Мультимедийные обучающие системы, 2010. http://sike.ru/articles/avtomatizirovannye-obuchayushchie-sistemy-kak-sredstvo-effektivnoi-podgotovki-spetsialistov
3. Бабанский Ю.К. Оптимизация учебно-воспитательного процесса - М.: Искусство, 1982. С.61-62.
4. Батищев В., Котова С.П., Скобелев П.О. «Мультиагентный подход для развития системы дистанционного обучения.
5. Глушань В.М., Марков В.В., Романов Р.М. «Компьютерные обучающие системы с адаптацией к психо-эмоциональному состоянию обучаемого», http://www.mgopu.ru/JOURNAL/15\_glushan.doc
6. Голенков В.В., Емельянов В.В., Тарасов В.Б. Виртуальные кафедры и интеллектуальные обучающие системы // Новости ИИ., 2001, №4. С.2-6.
7. Горбатюк Н.В., Божич В. И. «Интеллектуальная система компьютерного обучения»
8. Ильясов И.И. «Адаптационные процессы в обучении», // Издательство Московского университета, 1988 С 24-31
9. Кудрявцев В.Б., Вашик К., Алисейчик П.А., Кнап и др. Моделирование процесса обучения, М. 2004.
10. Кудрявцев В.Б., Вашик К., Строгалов А.С., Алисейчик П.А., Перетухин В.В. Компьютерная система обучения автоматного типа, М.: РГГ, 1996. С.111
11. Кудрявцев В.Б., Вашик К., Строгалов А.С., Алисейчик П.А., Перетухин В.В. Об автоматом моделировании процесса обучения, М.: РГГ, 1996. С.-10
12. Морев И. А. Образовательные информационные технологии. Часть 2. Педагогические измерения: Учебное пособие. – Владивосток: Изд-во Дальневост. ун-та, 2004. – 174 с.
13. Немцов А.Б. Использование автоматизированных обучающих систем на железнодорожном транспорте // Современные проблемы информатизации в непромышленной сфере и экономике: Сб. трудов. Вып. 10. Воронеж: Издательство "Научная книга", 2005. С.33-54.
14. Плещёв В.В. «Адаптивное образование в области программирования и баз данных: методология и методическое обеспечение» // Вестник УГТУ?УПИ, 2005. № 6
15. Поликарпова Н. И., Шалыто А. А. Автоматное программирование. 2008. — 167 с.:
16. Попов Э.В. Экспертные системы: решение неформализованных задач в диалоге с ЭВМ.- М.: Наука. 1987г.-288с
17. Хорстманн К.С., Корнелл Г. «Java 2. Библиотека профессионала, том 2. Тонкости программирования»
18. Хорстманн К.С., Корнелл Г. «Java 2. Библиотека профессионала. Том 1. Основы»
19. Цибульский Г.М., Герасимова Е.И., Ерошин В.В «Модели обучения автоматизированных обучающих систем» // Сетевой электронный научный журнал "Системотехника", 2004 г. http://systech.miem.edu.ru/2004/n2/Cibulskiy.htm
20. Шилдт Г. Полный справочник по Java, Вильямс, 2007г.
21. Сайт по автоматному программированию. http://is.ifmo.ru
22. «Система образования неэффективна без цифровых технологий. Речь Руперта Мердока на Саммите большой восьмерки», http://adindex.ru/publication/opinion/offtop/2011/05/27/66507.phtml
23. «Электронное обучение для учебных заведений», www.competentum.ru
24. Eckel B. Thinking in Java 2. Prentice-Hall. 2000г
25. Harel D., Pnueli A. On the development of reactive systems / In «Logic and Models of Concurrent Systems». NATO Advanced Study Institute on Logic and Models for Verification and Specification of Concurrent Systems. Springer Verlag, 1985. pp. 477–498.
26. Rasch, G. Probabilistic Models for Some Intelligence and Attainment Tests. With a Foreword and Afterword by B.D. Wright. The Univ. of Chicago Press. Chicago & London, 1980 . 199 рр.
27. Drozdek, Adam. "Data Structures and Algorithms in C++". Brook/Cole. Pacific Grove, CA. 2001. Second edition.

**Приложение 1** (Листинг программы).

Полная версия программы представлена в электронном виде.

**AnalyzeAnswer .java – Анализатор ответов.**

package tltsu.expertsystem.answeranalyzer;

import org.apache.log4j.Logger;

import java.util.Stack;

/\*\*

\* Class allows determine error in user's formula.

\* @author FADEEV M

\*/

public class AnalyzeAnswer

{

private static final Logger log = Logger.getLogger(AnalyzeAnswer.class);

/\*\*

\* This method build tree for two formulas and compare. Returned result with errors.

\* @param etalon correct formula

\* @param users formula that will be inspected

\* @return assessment and error is occurred errors

\* @throws Exception throws if error occurred in etalon formula

\*/

public static TestResult analyzeFormula(String etalon, String users) throws Exception

{

boolean teacherError = true;

FormulaTree etalonTree;

FormulaTree userTree;

try

{

etalonTree = TreeBuilder.build(etalon);

teacherError = false;

userTree = TreeBuilder.build(users);

}

catch (Exception e)

{

log.error("Error while parsing user/etalon formulas", e);

if (teacherError)

throw e;

else

return new TestResult(TestResult.BAD, e.getMessage());

}

FormulaTree mismatch = compareFormulas(etalonTree, userTree);

return createAnswerByMismatch(mismatch);

}

/\*\*

\* Compare trees by preorder traversal.

\* @param etalon etalon tree

\* @param user user tree

\* @return node that does not match

\*/

static FormulaTree compareFormulas(FormulaTree etalon, FormulaTree user)

{

Stack<FormulaTree> etalonNodes = new Stack<FormulaTree>();

Stack<FormulaTree> userNodes = new Stack<FormulaTree>();

etalonNodes.push(etalon);

userNodes.push(user);

FormulaTree currentEtalon;

FormulaTree currentUser;

while (!etalonNodes.isEmpty())

{

currentEtalon = etalonNodes.pop();

currentUser = userNodes.pop();

if (!currentEtalon.getValue().equals(currentUser.getValue()))

return currentUser;

FormulaTree rightEtalon = currentEtalon.right();

FormulaTree rightUser = currentUser.right();

FormulaTree leftEtalon = currentEtalon.left();

FormulaTree leftUser = currentUser.left();

if (rightEtalon == null && rightUser != null || rightEtalon != null && rightUser == null)

return rightUser;

if (leftEtalon == null && leftUser != null || leftEtalon != null && leftUser == null)

return rightUser;

if (leftEtalon == null && leftUser == null)

continue;

if (rightEtalon == null && rightUser == null)

continue;

if (!rightEtalon.getValue().equals(rightUser.getValue()))

{

if (isAssociated(currentEtalon))

{

if (rightEtalon.getValue().equals(leftUser.getValue()))

{

swap(rightUser, leftUser);

rightUser = currentUser.right();

leftUser = currentUser.left();

}

else

{

return rightUser;

}

}

else

return rightUser;

}

etalonNodes.push(rightEtalon);

userNodes.push(rightUser);

if (!leftEtalon.getValue().equals(leftUser.getValue()))

{

if (isAssociated(currentEtalon) && isAssociated(rightEtalon))

{

Stack<FormulaTree> subRight = new Stack<FormulaTree>();

subRight.push(rightUser);

FormulaTree search;

while (!leftEtalon.getValue().equals(leftUser.getValue()) && !subRight.empty())

{

search = subRight.pop();

if (search.left().getValue().equals(leftEtalon.getValue()))

{

swap(search.left(), leftUser);

leftUser = currentUser.left();

}

else if (search.right().getValue().equals(leftEtalon.getValue()))

{

swap(search.right(), leftUser);

leftUser = currentUser.left();

}

else

{

if (isAssociated(search.right()))

subRight.push(search.right());

if (isAssociated(search.left()))

subRight.push(search.left());

if (subRight.isEmpty())

return leftUser;

}

}

}

else

return leftUser;

}

userNodes.push(leftUser);

etalonNodes.push(leftEtalon);

}

if (!userNodes.empty())

throw new RuntimeException("Etalon stack is empty but, users - not");

return null;

}

/\*\*

\* This method for check whether the operation is associative.

\* @param tree operation

\* @return true is node (operation) associative, false otherwise.

\*/

static boolean isAssociated(FormulaTree tree)

{

return "\*".equals(tree.getValue()) || "+".equals(tree.getValue());

}

/\*\*

\* Allows swap 2 nodes.

\* @param a swapped node

\* @param b swapped node

\*/

static void swap(FormulaTree a, FormulaTree b)

{

a.swap(b);

}

/\*\*

\* This method for check whether the node is number.

\* @param operator node

\* @return true is node (operation) associative, false otherwise.

\*/

static boolean isNumber(String operator)

{

return TreeBuilder.isNumber(operator);

}

/\*\*

\* Method create result by node. If node is null, returned excellent result.

\* @param node node that is mismatch or null

\* @return result represented as {@link TestResult}

\*/

static TestResult createAnswerByMismatch(FormulaTree node)

{

String answer;

if (node == null)

return new TestResult();

else if (TreeBuilder.isFunction(node.getValue()))

answer = "Wrong function '" + node + "'";

else if (TreeBuilder.isOperator(node.getValue()))

answer = "Wrong operator '" + node + "'";

else if (TreeBuilder.isNumber(node.getValue()))

answer = "Wrong argument '" + node + "' in operator " + node.getParent();

else

answer = "mistake are not recognized";

return new TestResult(TestResult.BAD, "You are do mistake: " + answer);

}

}

**EduFSM.java – Конечный автомат используемый при обучении.**

package tltsu.expertsystem.fsm;

import org.apache.log4j.Logger;

import pt.fsm.FSM;

import pt.fsm.State;

import static tltsu.expertsystem.answeranalyzer.TestResult.BAD;

import static tltsu.expertsystem.answeranalyzer.TestResult.EXCELLENT;

import static tltsu.expertsystem.answeranalyzer.TestResult.NORMAL;

/\*\*

\* @author FADEEV

\*/

public class EduFSM implements Fsm

{

private static final Logger log = Logger.getLogger(EduFSM.class);

private FSM<States, Event> a;

public EduFSM()

{

a = createA();

}

// public void main(){}

public void handleEvent(Event event)

{

a.handleEvent(event);

}

public String getCurrentState()

{

return a.getCurrentState().getId().textualState;

}

enum States

{

/\*\*

\* increase theme and don't change level.

\*/

S\_U("0;1"),

/\*\*

\* dont change level and theme.

\*/

S\_S("0;0"),

/\*\*

\* increase level and theme.

\*/

U\_U("1;1"),

/\*\*

\* decrease level and don't change theme.

\*/

D\_S("-1;0"),

/\*\*

\* decrease level and increase theme.

\*/

D\_U("-1;1");

public final String textualState;

private States(String textualState)

{

this.textualState = textualState;

}

}

@SuppressWarnings("unchecked")

static private <E extends Object> FSM<States, Event> createA() {

return new FSM<States, Event>(

// STATE increase theme and don't change level.

new State<States, Event>

(States.S\_U) {

public void handleEvent()

{

Event e = e();

log.debug("handle event "+e);

if (e.lastResult == BAD)

next(States.S\_S);

else if (e.avgResults == BAD)

next(States.D\_U);

else if (e.eduTime == NORMAL && e.avgResults == EXCELLENT

&& (e.lastResult == NORMAL || e.lastResult == EXCELLENT))

next(States.U\_U);

else

next(States.S\_U);

}

public void enter() { log.debug("enter into "+States.S\_U.textualState); }

},

// STATE dont change level and theme.

new State<States, Event>

(States.S\_S) {

public void handleEvent()

{

Event e = e();

log.debug("handle event "+e);

if ( e.lastResult == EXCELLENT || e.lastResult == NORMAL && e.avgResults +e.eduTime >= EXCELLENT)

next(States.S\_U);

else

next(States.D\_S);

}

public void enter() { log.debug("enter into "+States.S\_S.textualState); }

},

// STATE increase level and theme.

new State<States, Event>

(States.U\_U) {

public void handleEvent()

{

log.debug("handle event "+e());

if ( e().lastResult == BAD)

next(States.S\_S);

else

next(States.S\_U);

}

public void enter() { log.debug("enter into "+States.U\_U.textualState); }

},

// STATE decrease level and don't change theme.

new State<States, Event>

(States.D\_S) {

public void handleEvent()

{

log.debug("handle event "+e());

if ( e().lastResult == EXCELLENT)

next(States.S\_U);

else

next(States.D\_S);

}

public void enter() { log.debug("enter into "+States.D\_S.textualState); }

},

// STATE decrease level and increase theme.

new State<States, Event>

(States.D\_U) {

public void handleEvent()

{

log.debug("handle event "+e());

if ( e().lastResult == BAD)

next(States.S\_S);

else

next(States.S\_U);

}

public void enter() { log.debug("enter into "+States.D\_U.textualState); }

// public void exit() { log.debug("exit3"); }

}

);

}

}

**Fsm.java – Интерфейс конечного автомата.**

package tltsu.expertsystem.fsm;

/\*\*

\* @author FADEEV

\*/

public interface Fsm

{

void handleEvent(Event event);

String getCurrentState();

}

**Utils.java – Вспомогательный класс.**

package tltsu.expertsystem.utils;

import org.apache.log4j.Logger;

import tltsu.expertsystem.\*;

import tltsu.expertsystem.AbstractUser;

import tltsu.expertsystem.MiddleUser;

import java.net.URL;

import java.util.LinkedList;

import java.util.List;

import java.util.Random;

import java.util.regex.Pattern;

/\*\*

\* @author FADEEV

\*/

public class Utils

{

private static final Logger log = Logger.getLogger(Utils.class);

public static final String YES = "YES";

public static final String NO = "NO";

private static List<Integer> maxQuestionInGroup = new LinkedList<Integer>();

public static AbstractUser createOrLoadUser(String name, String needCreate) throws SerializeException

{

if (needCreate.equals(YES))

return createOrLoadUser(name, true);

else if (needCreate.equals(NO))

return createOrLoadUser(name, false);

else

throw new IllegalArgumentException("Argument needCreate is not correct (YES/NO) = "+needCreate);

}

public static AbstractUser createOrLoadUser(String name, boolean needCreate) throws SerializeException

{

AbstractUser user;

if(name.equals("null") || name.isEmpty())

throw new IllegalArgumentException("User name is empty");

else if (Pattern.compile("\\W", Pattern.UNICODE\_CASE | Pattern.CASE\_INSENSITIVE).matcher(name).find())

throw new IllegalArgumentException("User name not correct = " + name);

if (needCreate)

{

log.debug("your user "+name+"are CREATED!");

user = new MiddleUser(EduProperties.COURSE, name);

}

else

{

user = AbstractUser.load(name);

log.debug("your user "+name+"are load! continue education with user characteristic"+user.getInfo());

}

return user;

}

/\*\*

\* Replace in patternCourse symbols "?" on arguments (args) step by step.

\* Example patternCourse= "material/sharph\_?\_?.html", args[0] = "2", args[1] = "hard" and main class located in c:/education/.

\* The invoke of this method return URL with path = file:///C:/education/material/sharph\_2\_hard.html

\* @param patternCourse a relative (relative from the main class) or absolute path

\* @param args arguments witch be replaced in patternCourse

\* @return url to file

\*/

public static URL getUrlByPattern(String patternCourse, String... args)

{

String resultPath = patternCourse;

for (String arg : args)

{

resultPath = resultPath.replaceFirst("\\?", arg);

}

if (resultPath == null)

throw new IllegalArgumentException("Argument pattern course can't be null.");

return Main.class.getResource(resultPath);

}

/\*\*

\* See {@link #getUrlByPattern(String, String[]) getUrlByPattern}

\*/

public static URL getUrlByPattern(String patternCourse, int... args)

{

return getUrlByPattern(patternCourse, convert(args));

}

/\*\*

\* Convert int array to string array.

\* @param converted array of int numbers

\* @return array of numbers in text representation

\*/

static String[] convert(int[] converted)

{

String[] result = new String[converted.length];

int i = 0;

for (int var :converted)

result[i++] = String.valueOf(var);

return result;

}

/\*\*

\* Return numbers in interval from 1 to {@link tltsu.expertsystem.EduProperties#MAX\_NUMBER\_QUESTION\_IN\_GROUP}

\*/

public static int getNextRandom()

{

if (maxQuestionInGroup.isEmpty())

{

for(int i=1; i<= EduProperties.MAX\_NUMBER\_QUESTION\_IN\_GROUP; i++)

maxQuestionInGroup.add(i);

}

Random rand = new Random();

Integer randVal;

do

{

randVal = rand.nextInt(EduProperties.MAX\_NUMBER\_QUESTION\_IN\_GROUP+1);

}

while (!maxQuestionInGroup.contains(randVal));

maxQuestionInGroup.remove(randVal);

return randVal;

}

public static int getNextRandom(int max)

{

if (maxQuestionInGroup.isEmpty())

{

for(int i=0; i< max; i++)

maxQuestionInGroup.add(i);

}

Random rand = new Random();

Integer randVal;

do

{

randVal = rand.nextInt(max);

}

while (!maxQuestionInGroup.contains(randVal));

maxQuestionInGroup.remove(randVal);

return randVal;

}

}

**AbstractUser.java – Класс представляющий абстрактного пользователя.**

package tltsu.expertsystem;

import org.apache.log4j.Logger;

import tltsu.expertsystem.answeranalyzer.TestResult;

import tltsu.expertsystem.fsm.Event;

import tltsu.expertsystem.ui.LectureHTML;

import tltsu.expertsystem.utils.SerializeException;

import tltsu.expertsystem.utils.Utils;

import java.io.\*;

import java.net.URL;

import java.util.LinkedList;

/\*\*

\* @author FADEEV

\*/

public abstract class AbstractUser implements Serializable

{

private static final Logger log = Logger.getLogger(AbstractUser.class);

private static final long serialVersionUID = 3778333413425545224L;

protected LinkedList<Integer> chapters;

protected LinkedList<Integer> levels;

protected LinkedList<Integer> chapterResults; // IN procent

protected ChapterTest chapterTest;

protected String course;

private int eduStep;

public final String userName;

public AbstractUser(String pathToXml, String name)

{

userName = name;

eduStep = 0;

chapters = new LinkedList<Integer>();

levels = new LinkedList<Integer>();

chapterResults = new LinkedList<Integer>();

setCourse(pathToXml);

chapters.addLast(0);

levels.addLast(0);

}

public int getEduStep()

{

return eduStep;

}

public String getCourse()

{

return course;

}

public void setCourse(String course)

{

this.course = course;

}

public abstract void changeEduState(String state);

public abstract Event calculateEduState();

public abstract boolean isEduComplete();

public void processChapterTestResult(ChapterTest chapterTest)

{ // тут нахождение средней результатов теста : находим среднее по всем групповым тестам.

Double partResult = 0.0;

for (int part = 0; part < chapterTest.getCurrentPart(); part++)

{

partResult += chapterTest.getPartResult(part);

log.info("step"+part+"partResult increased = "+partResult);

}

partResult /= chapterTest.getCurrentPart();

if (partResult > 90)

chapterResults.addLast(TestResult.EXCELLENT);

else if (partResult > 60)

chapterResults.addLast(TestResult.NORMAL);

else

chapterResults.addLast(TestResult.BAD);

log.info("set Current result = "+ chapterResults.getLast()+ " where % ~= "+ partResult);

}

public void teach() throws Exception

{

log.debug("now teach! material from chapter#" + chapters.getLast() + " with level#" + levels.getLast());

// String path = getCourse().replaceFirst("\\?", String.valueOf( getCurrentLevel()));

// path = path.replaceFirst("\\?", String.valueOf(chapters.getLast()));

// URL url = this.getClass().getResource(path);

URL url = Utils.getUrlByPattern("material/"+getCourse(), levels.getLast(), chapters.getLast());

try

{

LectureHTML dialog = new LectureHTML("C sharp lection#"+ chapters.getLast(), url);

dialog.showLecture();

}

catch (Exception e)

{

log.error("error while show lecture! where URL="+ url + ". Rollback edu. ", e);

levels.remove(levels.size()-1);

chapters.remove(chapters.size()-1);

throw e;

}

}

public void testCurrentChapter() throws Exception

{

// if (chapterTest != null)

// throw new RuntimeException("Why chaptertest are keep saved ? this instance will be nullable in finish work method test");

chapterTest = new ChapterTest(getCourse(), chapters.getLast(), levels.getLast());

while (!chapterTest.isTestComplete()) // ходим по всем частям

{

chapterTest.nextPart();

chapterTest.testCurrPart(); // в testCurrPart ходим по вопросам пока не ответим. там же считаем результат.

// SHOW error in userAnswer if exist

// chapterTest.mistakesWorking();

}

processChapterTestResult(chapterTest);

eduStep++;

}

/\*\*

\* serialize to project directory in file "&lt;userName&gt;.usf"

\*/

public void save() throws SerializeException // saved to project directory

{

try

{

FileOutputStream fos = new FileOutputStream(userName + ".usf");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(this);

oos.flush();

oos.close();

log.trace("Successfully serialize user:"+getInfo()+" to " + userName + ".usf");

}

catch (IOException e)

{

log.error("Some error while serialize user", e);

throw new SerializeException("cant save user "+userName, e);

}

}

/\*\*

\* Deserialize from project directory file "&lt;userName&gt;.usf"

\* @param userName users name searched in directory by him

\* @return instanse of deserialized object

\*/

public static AbstractUser load(String userName) throws SerializeException

{

try

{

FileInputStream fis = new FileInputStream(userName + ".usf");

ObjectInputStream oin = new ObjectInputStream(fis);

return (AbstractUser) oin.readObject();

}

catch (Exception e)

{

log.error("Some error while deserialize user", e);

throw new SerializeException("Can't load user "+userName, e);

}

}

public String getInfo()

{

return "Name-"+userName+" Chapters:"+chapters +" Levels:" + levels + " Results"+ chapterResults + " Course" + course +

" Edu step"+ eduStep ;

}

}

**ChapterTest.java – Класс представляющий сущность теста одной темы.**

package tltsu.expertsystem;

import org.apache.log4j.Logger;

import tltsu.expertsystem.answeranalyzer.TestResult;

import tltsu.expertsystem.ui.InputQuestion;

import tltsu.expertsystem.ui.TestQuestion;

import tltsu.expertsystem.ui.SimpleLecture;

import tltsu.expertsystem.utils.Utils;

import java.io.Serializable;

import java.util.ArrayList;

/\*\*

\* @author FADEEV

\*/

public class ChapterTest implements Serializable

{

private static final long serialVersionUID = 374520639802143839L;

private static final Logger log = Logger.getLogger(ChapterTest.class);

int level;

int chapter;

protected ArrayList<Double> results;

protected Integer testParts;

protected ArrayList<Integer> testQuestion;

ConfigurationOfChapterTest chapterTestConfig;

public ChapterTest(String course, int chapter, int level) throws Exception

{

this.chapter = chapter;

this.level = level;

results = new ArrayList<Double>();

testParts = 0;

// testParts.add(0);

testQuestion = new ArrayList<Integer>();

// testQuestion.add(0);

// initialize config, parse count of PARTS

chapterTestConfig = ConfigurationOfChapterTest.load(course, level ,chapter);

}

@Deprecated

public Double getCurrentPartResult()

{

return results.get(results.size() - 1);

}

public void setCurrentPartResult(Double result)

{

this.results.add(result);

}

public Double getPartResult(int part)

{

return results.get(part);

}

public int getCurrentPart()

{

return testParts;

}

public void nextPart()

{

testParts += 1;

}

public boolean isTestComplete()

{ // getCurrentPart()+1 becouse currentParn begining with 0.

return getCurrentPart()+1 >= chapterTestConfig.getCountOfParts() /\*|| или 2 ответа зафейлины или другие "ништянки"\*/;

}

private void mistakesWorking(TestResult questionResult)

{

// System.out.println(questionResult.error != null +"||"+ !questionResult.error.isEmpty());

if(questionResult.error != null && !questionResult.error.isEmpty() && !questionResult.error.equals("null"))

new SimpleLecture("Correction of mistakes", questionResult.error).showLecture();

}

public void testCurrPart() throws Exception

{ // i - need for account of attempts to answer a part.

int i = 0;

// List<TestResult> resultQuestion = new LinkedList<TestResult>();

do

{

// 1) достаём данные для выдачи - она включает в себя данные о конкретном вопросе

// int maxAvailableInCurrentPart = chapterTestConfig.getCountOfQuestion(getCurrentPart());

TestQuestion question = parseQuestionXML(Utils.getNextRandom(/\*maxAvailableInCurrentPart\*/));

// 2.a) отдаём вопрос к UI -рисуем вопрсо и пишем ответ в нутрях класса

question.showLecture();

// 2.b) забираем у UI ответ и сравниваем его с эталоном (который вытащили из xml) с помощью анализатора и итоге получаем testResult

TestResult questionResult = question.getTestResult();

// resultQuestion.add(questionResult);

mistakesWorking(questionResult);

i++;

if (questionResult.result != TestResult.BAD )

break;

}

while (i < chapterTestConfig.getCountOfQuestion(getCurrentPart()));

double result = (100 + 30 \* (i-1) ) / i;

setCurrentPartResult(result);

}

private TestQuestion parseQuestionXML(Integer questionNumber)

{

log.debug("delivery question: level#"+level+" chapter#"+chapter+" part#"+getCurrentPart()+" question#"+questionNumber);

ConfigurationOfChapterTest.Pair pair = chapterTestConfig.getQuestionWithAnswer(getCurrentPart(), questionNumber);

return new InputQuestion("Question: "+pair.question, pair.answer);

// return new InputQuestion("Question: level#"+level + " chapter#"+chapter + " part#"+getCurrentPart()+ " number#"+questionNumber, "3\*y+y^2");

}

}

**User.java – Класс представляющий сущность пользователя.**

package tltsu.expertsystem;

import java.util.LinkedList;

import org.apache.log4j.Logger;

import tltsu.expertsystem.answeranalyzer.TestResult;

import tltsu.expertsystem.fsm.Event;

/\*\*

\* @author FADEEV

\*/

public class User extends AbstractUser

{

private static final Logger log = Logger.getLogger(User.class);

private static final long serialVersionUID = 3957424036423506327L;

// private ArrayList<Long> teachTime;

private LinkedList<Long> testTimes;

public User(String pathToMaterial, String name)

{

super(pathToMaterial, name);

// teachTime = new ArrayList<Long>();

testTimes = new LinkedList<Long>();

}

@Override

public void changeEduState(String state)

{

String[] characteristic = state.split(";");

int resultLevel = Integer.parseInt(characteristic[0]) + levels.getLast();

if (resultLevel>EduProperties.MAX\_LEVEL)

{

log.debug("resultLevel is too big ("+resultLevel+") for this configuration, use maximum level = "+EduProperties.MAX\_LEVEL);

resultLevel = EduProperties.MAX\_LEVEL;

}

else if (resultLevel < EduProperties.MIN\_LEVEL)

{

log.debug("resultLevel is too small ("+resultLevel+") for this configuration, use minimum level = "+EduProperties.MIN\_LEVEL);

resultLevel = EduProperties.MIN\_LEVEL;

}

int resultChapter = Integer.parseInt(characteristic[1]) + chapters.getLast();

levels.addLast(resultLevel);

chapters.addLast(resultChapter);

log.trace("set (add to list) new level =" + resultLevel);

log.trace("set (add to list) new chapter =" + resultChapter);

}

@Override

public Event calculateEduState()

{

if (getEduStep() == 0)

{

log.info("on first step not need calculate. All results = 1, need for delivery next chapter");

return new Event(TestResult.NORMAL, TestResult.NORMAL, TestResult.NORMAL);

}

double dynamicResult=0;

int testTime;

log.info("calculate new edu state");

if (testTimes.size() == 0) // for initial state (first iteration "while" )

{

log.info("testTime = TestResult.NORMAL (1), because testTimes.size() = 0");

testTime = TestResult.NORMAL;

}

else if (testTimes.getLast() <= EduProperties.NORMAL\_TEST\_TIME)

{

log.info("testTime = TestResult.NORMAL (1), because last testTime ("+testTimes.getLast()

+") <= EduProperties.NORMAL\_TEST\_TIME ("+EduProperties.NORMAL\_TEST\_TIME+")");

testTime = TestResult.NORMAL;

}

else

{

log.info("testTime = TestResult.BAD (0), because last testTime ("+testTimes.getLast()

+") > EduProperties.NORMAL\_TEST\_TIME ("+EduProperties.NORMAL\_TEST\_TIME+")");

testTime = TestResult.BAD;

}

for (Integer chResult : chapterResults)

dynamicResult += chResult;

dynamicResult/=chapterResults.size();

log.info(" dynamicResult = "+dynamicResult + " and round = "+Math.round(dynamicResult));

Event resultEvent = new Event(testTime, chapterResults.getLast(), (int) Math.round(dynamicResult));

log.info("Total event - "+resultEvent.toString());

return resultEvent;

}

@Override

public boolean isEduComplete()

{

return chapters.getLast() == EduProperties.MAX\_CHAPTER && chapterResults.getLast() >= TestResult.NORMAL;

}

public void testCurrentChapter() throws Exception

{

long startTestTime = System.currentTimeMillis();

super.testCurrentChapter();

testTimes.addLast(System.currentTimeMillis() - startTestTime);

}

public String getInfo()

{

return super.getInfo() + " Test time:" + testTimes;

}

}

**ConfigurationOfChapterTest.java – Класс представляющий тест по одной главе .**

package tltsu.expertsystem;

import org.apache.log4j.Logger;

import tltsu.expertsystem.utils.SerializeException;

import tltsu.expertsystem.utils.Utils;

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* @author FADEEV

\*/

public class ConfigurationOfChapterTest implements Serializable

{

private static final long serialVersionUID = -7104947342610224018L;

private static final Logger log = Logger.getLogger(Utils.class);

private List<List<Pair>> parts = new ArrayList<List<Pair>>();

public Pair getQuestionWithAnswer(int part, int question)

{

return parts.get(part).get(question);

}

public int getCountOfParts()

{

return parts.size();

}

public int getCountOfQuestion(int part)

{

return parts.get(part).size();

}

public void addPair(String question, String answer, int part)

{

List<Pair> currentPart;

try

{

currentPart = parts.get(part);

}

catch (IndexOutOfBoundsException e)

{

currentPart = new ArrayList<Pair>();

parts.add(part, currentPart);

}

currentPart.add(new Pair(question, answer));

}

public static ConfigurationOfChapterTest load(String course, int level, int chapter) throws SerializeException

{

try

{

// URL path = Utils.getUrlByPattern(course, level, chapter);

String path = course.substring(0, course.indexOf("."));

// path = path.replaceFirst("\\?", String.valueOf(level)).replaceFirst("\\?", String.valueOf(chapter)) + ".cct";

InputStream is = Utils.getStreamByPattern("material/test/"+path+".cct", String.valueOf(level), String.valueOf(chapter));

ObjectInputStream oin = new ObjectInputStream(is);

return (ConfigurationOfChapterTest) oin.readObject();

}

catch (Exception e)

{

log.error("Some error while deserialize config of chapter test : course ="+course+" level ="+level+ " chapter=" +chapter, e);

throw new SerializeException("Can't load config of chapter test", e);

}

}

public void save(String course, int level, int chapter) throws SerializeException // saved to project directory

{

try

{

String path = course.substring(0, course.indexOf("."));

path = path.replaceFirst("\\?", String.valueOf(level)).replaceFirst("\\?", String.valueOf(chapter)) + ".cct";

FileOutputStream fos = new FileOutputStream( new File(path));

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(this);

oos.flush();

oos.close();

log.trace("Successfully serialize config of chapter test " +path+ ".usf");

}

catch (Exception e)

{

log.error("Some error while deserialize config of chapter test : course ="+course+" level ="+level+ " chapter=" +chapter, e);

throw new SerializeException("cant save config of chapter test ", e);

}

}

public class Pair implements Serializable

{

private static final long serialVersionUID = -5833210109861521414L;

final public String question;

final public String answer;

private Pair(String question, String answer)

{

this.question = question;

this.answer = answer;

}

public String toString()

{

return question+"#"+answer;

}

}

}

**EduProperties.java – Класс представляющий глобальные конфигурации системы .**

package tltsu.expertsystem;

import org.apache.log4j.Logger;

import java.io.\*;

import java.util.Properties;

/\*\*

\* @author FADEEV

\*/

public class EduProperties

{

private static final String MAX\_CHAPTER\_STR = "MAX\_CHAPTER";

private static final String MAX\_LEVEL\_STR = "MAX\_LEVEL";

private static final String NORMAL\_TEACH\_TIME\_STR = "NORMAL\_TEACH\_TIME";

private static final String NORMAL\_TEST\_TIME\_STR = "NORMAL\_TEST\_TIME";

private static final String TEACHER\_NAME\_STR = "TEACHER\_NAME";

private static final String COURSE\_STR = "COURSE";

private static final String MAX\_NUMBER\_QUESTION\_IN\_GROUP\_STR = "MAX\_NUMBER\_QUESTION\_IN\_GROUP\_STR";

private static final Logger log = Logger.getLogger(EduProperties.class);

private static Properties properties;

static

{

properties = new Properties();

try

{

InputStream os = new DataInputStream(new FileInputStream("global.property"));

properties.loadFromXML(os);

log.debug("Successfully loaded configuration");

testLogEduProperties();

}

catch (Exception e)

{

log.error("Can't load configuration, use default.", e);

properties.setProperty(MAX\_CHAPTER\_STR, "10");

properties.setProperty(MAX\_LEVEL\_STR, "2");

properties.setProperty(NORMAL\_TEACH\_TIME\_STR, "30");

properties.setProperty(NORMAL\_TEST\_TIME\_STR, "10");

properties.setProperty(MAX\_NUMBER\_QUESTION\_IN\_GROUP\_STR, "4");

properties.setProperty(COURSE\_STR, "diff\_?\_?.htm");

try

{

OutputStream os = new DataOutputStream(new FileOutputStream("global.property"));

properties.storeToXML(os, null);

}

catch (Exception e1)

{

log.error("Can't save configuration, in next run again created default properties.", e);

}

}

}

public static final int MAX\_CHAPTER = Integer.parseInt((String) properties.get(MAX\_CHAPTER\_STR)) ;

public static final int MAX\_LEVEL = Integer.parseInt((String) properties.get(MAX\_LEVEL\_STR));

public static final int MIN\_LEVEL = 0;

public static final int NORMAL\_TEACH\_TIME = Integer.parseInt((String) properties.get(NORMAL\_TEACH\_TIME\_STR));

public static final int NORMAL\_TEST\_TIME = Integer.parseInt((String) properties.get(NORMAL\_TEST\_TIME\_STR)) \* 1000 \* 60;

public static final int MAX\_NUMBER\_QUESTION\_IN\_GROUP = Integer.parseInt((String) properties.get(MAX\_NUMBER\_QUESTION\_IN\_GROUP\_STR));

public static final String COURSE =(String) properties.get(COURSE\_STR) ;

public static final String TEACHER\_NAME =(String) properties.get(TEACHER\_NAME\_STR) ;

public static void testLogEduProperties()

{

log.error("MAX\_CHAPTER int ="+EduProperties.MAX\_CHAPTER);

log.error("MAX\_LEVEL int ="+EduProperties.MAX\_LEVEL);

log.error("MIN\_LEVEL int ="+EduProperties.MIN\_LEVEL);

log.error("NORMAL\_TEACH\_TIME int ="+EduProperties.NORMAL\_TEACH\_TIME);

log.error("NORMAL\_TEST\_TIME int ="+EduProperties.NORMAL\_TEST\_TIME);

log.error("MAX\_NUMBER\_QUESTION\_IN\_GROUP int ="+EduProperties.MAX\_NUMBER\_QUESTION\_IN\_GROUP);

log.error("COURSE ="+EduProperties.COURSE);

log.error("TEACHER\_NAME int ="+EduProperties.TEACHER\_NAME);

}

}

**EduSystem.java – Класс позволяющий запускать процесс обучения.**

package tltsu.expertsystem;

import org.apache.log4j.Logger;

import tltsu.expertsystem.fsm.EduFSM;

import tltsu.expertsystem.fsm.Event;

import tltsu.expertsystem.fsm.Fsm;

import tltsu.expertsystem.ui.AbstractDialog;

import tltsu.expertsystem.ui.AbstractQuestion;

import tltsu.expertsystem.ui.SimpleLecture;

import tltsu.expertsystem.ui.StartWindow;

import tltsu.expertsystem.utils.SerializeException;

import tltsu.expertsystem.utils.Utils;

/\*\*

\* @author FADEEV

\*/

public class EduSystem

{

private static final Logger log = Logger.getLogger(EduSystem.class);

static private final int INDEX\_NAME = 0;

static private final int INDEX\_NEED\_CREATE = 1;

static void prepareAndRunEDU()

{

log.trace("Now start EDU System");

AbstractQuestion needCreateUser = new StartWindow();

needCreateUser.showLecture();

AbstractUser user;

String[] userInput = needCreateUser.getUserAnswer().split("#");

log.trace("witch user and are need create user? " + needCreateUser.getUserAnswer());

try

{

user = Utils.createOrLoadUser(userInput[INDEX\_NAME], userInput[INDEX\_NEED\_CREATE]);

}

catch (SerializeException e)

{

log.error("Error while create/load user, is need create "+userInput[INDEX\_NEED\_CREATE] + " with name = "+userInput[INDEX\_NAME],e);

AbstractDialog error = new SimpleLecture("Error", e.getMessage()+"\nEducation is over");

error.setResizable(false);

error.showLecture();

return;

}

try

{

EduSystem.runEdu(user);

}

catch (Exception e)

{

log.error("Some error.", e);

}

finally

{

log.info("Finally try save user");

try

{

user.save();

}

catch (SerializeException e)

{

log.error("Can't save user");

AbstractDialog error = new SimpleLecture("Error", "Can't save user.");

error.setResizable(false);

error.showLecture();

}

}

}

/\*\*

\* use default {@link tltsu.expertsystem.fsm.EduFSM}

\* @param user

\* @throws Exception

\*/

public static void runEdu(AbstractUser user) throws Exception

{

runEdu(new EduFSM(), user);

}

public static void runEdu(Fsm education, AbstractUser user) throws Exception

{

while (!user.isEduComplete()) //вайл НЕ закончено обучение

{

log.debug("user state after check isEduComplete "+user.getInfo());

Event eduState = user.calculateEduState();

education.handleEvent(eduState); // посылаем результат тестирования на автомат (level up/down/hold : theme up/down/hold )

String newState = education.getCurrentState(); // вернёт строку вида "0;1" или ??"0;0;0;1"

user.changeEduState(newState);

log.debug("user state after changeEduState "+user.getInfo());

user.teach();

log.debug("user state after teach "+ user.getInfo());

user.testCurrentChapter();

log.debug("user state after testCurrentChapter "+ user.getInfo());

}

SimpleLecture dialog = new SimpleLecture("Education", "Education is complete!");

dialog.showLecture();

log.info("Education is complite!");

}

}

**FormulaTree.java – Класс представляющий собой дерево позволяющиее хранить формулы в удобной структуре.**

package tltsu.expertsystem.answeranalyzer;

import org.apache.log4j.Logger;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* @author FADEEV

\*/

public class FormulaTree

{

private static final Logger log = Logger.getLogger(FormulaTree.class);

private String value;

private ArrayList<FormulaTree> children;

private FormulaTree parent ;

public FormulaTree(String vertexValue)

{

value = vertexValue;

parent = null;

children = new ArrayList<FormulaTree>();

}

public void addVertex(String value)

{

FormulaTree child = new FormulaTree(value);

this.children.add(child);

child.parent = this;

}

public void addVertex(FormulaTree value)

{

value.parent = this;

this.children.add(value);

}

String getValue()

{

return value;

}

public String toString ()

{

return getValue();

}

public List<FormulaTree> getChild()

{

return children;

}

public static void printTree (FormulaTree tree)

{

for (int i =0; i<tree.getLevel();i++)

System.out.print(" ");

System.out.print(tree);

for (int i =0; i<tree.getLevel();i++)

System.out.print(" ");

System.out.println(tree.parent !=null ? " ("+tree.parent+") " :" (root)");

for (FormulaTree child : tree.getChild())

printTree(child);

}

//

int getLevel ()

{

int result = 0 ;

FormulaTree tree=this;

while (tree.parent!=null)

{

tree= tree.parent;

result++;

}

return result;

}

public FormulaTree right()

{

if(children.size()>=2)

return children.get(1);

else

return null;

}

public FormulaTree left()

{

if(children.size()>=1)

return children.get(0);

else

return null;

}

void swap(FormulaTree tree)

{

log.debug("Swap "+tree + " and "+this);

// FormulaTree buff = this;

int a = tree.parent.children.indexOf(tree);

int b = this.parent.children.indexOf(this);

tree.parent.children.set(a, this);

this.parent.children.set(b, tree);

FormulaTree buffParent = tree.parent;

tree.parent = this.parent ;

this.parent = buffParent;

}

public FormulaTree getParent()

{

return parent;

}

}

**TreeBuilder.java – Класс позволяющий строить дерево по формуле представленной строкой.**

package tltsu.expertsystem.answeranalyzer;

import org.apache.log4j.Logger;

import java.text.ParseException;

import java.util.Collections;

import java.util.Stack;

import java.util.StringTokenizer;

/\*\*

\* @author FADEEV

\*/

public class TreeBuilder

{

private static final Logger log = Logger.getLogger(TreeBuilder.class);

private static final String[] FUNCTIONS = {"abs", "acos", "arg", "asin", "atan", "conj", "cos", "cosh", "exp",

"imag", "log", "neg", "pow", "real", "sin", "sinh", "sqrt", "tan", "tanh"};

private static final String OPERATORS = "+-\*/^";

private static final String SEPARATOR = ",";

protected static Stack<String> parse(String expression) throws ParseException

{

Stack<String> stack = new Stack<String>();

Stack<String> result = new Stack<String>();

expression = expression.replace(" ", "").replace("°", "\*" + Double.toString(Math.PI) + "/180").replace("(-",

"(0-").replace(",-", ",0-").replace("(+", "(0+").replace(",+", ",0+");

if (expression.charAt(0) == '-' || expression.charAt(0) == '+')

{

expression = "0" + expression;

}

StringTokenizer stringTokenizer = new StringTokenizer(expression, OPERATORS + SEPARATOR + "()", true);

while (stringTokenizer.hasMoreTokens())

{

String token = stringTokenizer.nextToken();

if (isNumber(token))

{

result.push(token);

}

else if (isFunction(token))

{

stack.push(token);

}

else if (isSeparator(token))

{

while (!isOpenBracket(stack.lastElement()))

{

if (stack.size() == 1)

throw new ParseException("Open brace or separator are missed", 0);

result.push(stack.pop());

}

}

else if (isOperator(token))

{

while (!stack.empty() && isOperator(stack.lastElement()) && getPrecedence(token) <= getPrecedence(

stack.lastElement()))

{

result.push(stack.pop());

}

stack.push(token);

}

else if (isOpenBracket(token))

{

stack.push(token);

}

else if (isCloseBracket(token))

{

while (!stack.empty() && !isOpenBracket(stack.lastElement()))

{

result.push(stack.pop());

}

if (stack.empty())

throw new ParseException("Error parsing, input expression missed bracket", 0);

stack.pop();

if (!stack.empty() && isFunction(stack.lastElement()))

{

result.push(stack.pop());

}

}

else

{

log.error("Unrecognized token");

throw new ParseException("Unrecognized token", 0);

}

}

while (!stack.empty())

{

String operation = stack.pop();

if (isOpenBracket(operation))

{

log.error("Error parsing, input expression contains not closed bracket");

throw new ParseException("Error parsing, input expression contains not closed bracket", 0);

}

else

result.push(operation);

}

return result;

}

public static FormulaTree build(String expr) throws Exception

{

Stack<String> infix;

try

{

infix = parse(expr);

}

catch (ParseException e)

{

log.error("Error while parsing");

throw new Exception("Parsing Exception. \nError occured while parsing \"" + expr + "\"", e);

}

Collections.reverse(infix);

Stack<FormulaTree> stack = new Stack<FormulaTree>();

try

{

while (!infix.empty())

{

String token = infix.pop();

if (isNumber(token))

{

stack.push(new FormulaTree(token));

}

else if (isFunction(token) || isOperator(token))

{

int argumentCount = argumentCount(token);

if (stack.size() < argumentCount)

{

log.error("Build Exception. Too few arguments for " + token);

throw new Exception("Build Exception. Too few arguments for " + token);

}

FormulaTree operation = new FormulaTree(token);

for (int i = 0; i < argumentCount; i++)

operation.addVertex(stack.pop());

stack.push(operation);

}

}

if (stack.size() == 1)

{

return stack.pop();

}

else

{

log.error("Build Exception. Wrong input expression " + expr + "\"");

throw new Exception("Build Exception. Wrong input expression \"" + expr + "\"");

}

}

catch (Exception e)

{

log.error("Erorr while parsing");

throw new Exception("Build Exception. \"" + expr + "\", bad parsing or input expression", e);

}

}

protected static boolean isNumber(String token)

{

if ("x".equalsIgnoreCase(token))

return true;

if ("y".equalsIgnoreCase(token))

return true;

try

{

Double.parseDouble(token);

}

catch (Exception e)

{

return false;

}

return true;

}

protected static boolean isFunction(String token)

{

for (String item : FUNCTIONS)

{

if (item.equals(token))

{

return true;

}

}

return false;

}

protected static boolean isSeparator(String token)

{

return SEPARATOR.equals(token);

}

protected static boolean isOpenBracket(String token)

{

return "(".equals(token);

}

protected static boolean isCloseBracket(String token)

{

return ")".equals(token);

}

static boolean isOperator(String token)

{

return OPERATORS.contains(token);

}

static byte getPrecedence(String token)

{

if ("+".equals(token) || "-".equals(token))

return 1;

else if ("^".equals(token))

return 3;

else

return 2;

}

static byte argumentCount(String operator) throws Exception

{

if (isOperator(operator))

return 2;

else if ("!".equals(operator))

return 1;

else if ("pow".equals(operator))

return 2;

else if (isFunction(operator))

return 1;

log.error("no such operator \"" + operator + "\"");

throw new Exception("no such operator \"" + operator + "\"");

}

}