УДК 519.6:519.81

ПРОГРАММНАЯ ОБОЛОЧКА ПОДДЕРЖКИ И СИНТЕЗА РАЦИОНАЛЬНЫХ РЕШЕНИЙ

И. Е. Воронина, Е. Ю. Митрофанова

Воронежский государственный университет

Поступила в редакцию 10.10.2009

Аннотация. Рассматривается задача создания программной оболочки поддержки и синтеза рациональных решений с возможностью динамического пополнения набора методов. Программный интерфейс разработан в соответствии с основными эргономическими требованиями. Работоспособность программной оболочки проиллюстрирована реализацией двух методов поддержки и синтеза рациональных решений.

Ключевые слова: принятие и синтез рациональных решений, метод анализа иерархий (МАИ), метод ассоциаций, динамическая загрузка методов.

Abstract. The article is devoted to the task of development of the software shell for supporting and synthesis rational decisions

Key words: supporting and synthesis rational decisions, method of the analysis of hierarchies, method of associations, dynamic loading.

ВВЕДЕНИЕ

Для проведения системного анализа любых видов человеческой деятельности, связанной с принятием управленческих решений, используются различные методы поддержки принятия и синтеза решений.

Задача принятия решений (ЗПР) — одна из самых распространенных в любой предметной области. ЗПР заключается в выборе одной или нескольких лучших альтернатив из некоторого первоначального набора. Для того чтобы сделать такой выбор правильно и как можно ближе к идеальному результату, необходимо четко определить цель и критерии, по которым будет проводиться оценка набора альтернатив. Выбор метода решения такой задачи зависит от количества и качества предоставленной информации. Методы принятия, планирования и синтеза решений основываются на применении знаний (в частности, системы предпочтений) лица или коллектива лиц, то есть коллективные решения могут приниматься лишь в связи с какими-либо целями деятельности *лица, принимающего решение* (ЛПР). Под *целью* понимается в широком смысле идеальное представление желаемого состояния или результата деятельности. *Вариант решения* (или просто *решение*) — это возможный способ достижения поставленной цели. Варианты должны быть взаимоисключающими или альтернативными, поэтому вариант решения по-другому называется *альтернативой*. Реализация каждой альтернативы приводит к наступлению некоторых последствий (исходов), анализ и оценка которых осуществляется по одному критерию или нескольким, образующим множество критериев, однозначно характеризующих свойства альтернатив [1].

Принятие решений — особый вид целенаправленной деятельности, включающий определение целей, постановку задачи принятия решений и саму процедуру принятия решений — выбор одной из имеющихся альтернатив.

Для практического использования метода необходимо изучить математический аппарат, который лежит в его основе. Во многих случаях выполнение вычислений может занять достаточно много времени. Часто это весьма трудоемкий процесс, требующий концентрации внимания на деталях.

Процедуры выявления знаний, предпочтения самого ЛПР, настолько сложны и неоднозначны, что требуют участия консультанта в процессе выбора решения из множества пред-

ставленных альтернатив. Консультант, как правило, должен полностью владеть всей информацией о методах принятия и синтеза решений, приемлемых при различных критериях, альтернатив, шкалах критериев, типах оценок и т. п. Привлекаемые к процессу решения поставленной задачи специалисты помогают ЛПР более четко разобраться в сложившейся ситуации выбора решений, обучают его применяемым методам. Опыт консультанта обеспечивает целенаправленность размышлений ЛПР. Все это дает пользователям возможность синтеза и выявления наиболее обоснованных вариантов из всего множества допустимых.

Выявление данных, знаний и системы предпочтений ЛПР для решения задачи осуществляется путем сбора экспертной информации. Разработка различных анкет для вариантов задач принятия и синтеза решений невозможна. Следовательно, требуется постоянное участие консультанта, направляющего последовательность рассуждений ЛПР в процессе сбора экспертной информации, что, в свою очередь, может привести к нарушению принципов конфиденциальности и необходимой документированности информации. Если решение, выбранное из множества альтернатив предлагаемым образом, является неудовлетворительным для ЛПР, консультант не имеет никакой возможности восстановить процедуру принятия решения. Это, в свою очередь, приводит к тому, что невозможно обосновать справедливость полученного решения. Создание инструментальных средств, которые программно реализуют механизм принятия и синтеза решений и берут на себя роль консультанта, позволяет обеспечить как конфиденциальность принятия решения, так и рациональное распределение функций между пользователем и ЭВМ.

Следует заметить, что множество различных методов принятия решения достаточно велико и, кроме того, открыто. Реализовать все известные на текущий момент методы нет смысла. Поэтому более разумно разработать программную оболочку, поддерживающую механизм добавления новых методов, реализованных по мере необходимости.

Автоматизация технологии принятия решений призвана, прежде всего, перевести все соответствующие действия в явную, осознанную форму, как с целью минимизации возможных ошибок, так и с целью применения специально разработанных приемов для достижения наилучшего результата. В исследовательской работе не всегда обязательно детальное изучение используемого метода, а важны лишь экспертная информация и конечный результат, в то время как в учебном процессе чаще всего подразумевается не только использование, но и изучение самих методов.

Все вышесказанное обуславливает актуальность создания программной оболочки, реализующей различные методы поддержки принятия решения, с возможностью пополнения набора методов.

Для демонстрации работы оболочка дополнена реализациями конкретных методов: метода анализа иерархий (МАИ) [2] и метода ассоциаций.

1. ОСОБЕННОСТИ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Как правило, коммерческие системы принятия решения базируются только на использовании одного из методов принятия решений, что не является достаточно удобным при необходимости провести исследовательскую работу с использованием более чем одного метода. В случае, когда необходимо воспользоваться сразу несколькими методами, исследователю придется оперировать набором программ, реализующими эти методы и обладающими различными интерфейсами. Это не только ухудшает зрительное восприятие, но и увеличивает время получения необходимого результата. Отсутствует возможность непосредственного изучения самих методов, возможно лишь их использование.

Особенностью реализованной программной оболочки является то, что пользователь может динамически добавлять в проект модули реализаций новых методов. Для этого используется механизм динамической загрузки произвольных классов.

Динамическая загрузка произвольных классов — возможность расширения и изменения функциональности приложения без его перекомпиляции.

В различных языках подобный механизм реализуется по-своему. Например, он поддерживается в Java и реализуется классом Class. В других языках, типа C++ или Delphi, аналогичных целей можно достичь, используя специальные средства конкретной операционной

системы (типа загрузки dll в Windows функцией loadLibrary()).

Существуют два метода работы с DLL:

1. Привязка DLL к программе. Это наиболее простой и легкий метод для использования функций, импортируемых из DLL. Однако (и на это следует обратить внимание), этот способ имеет очень весомый недостаток — если библиотека, которую использует программа, не будет найдена, то программа просто не запустится, выдавая ошибку и сообщая о том, что ресурс DLL не найден. А поиск библиотеки будет вестись в текущем каталоге, в каталоге программы, в каталоге WINDOWS\SYSTEM, и т.д. Например:

Implementation

function FunctionName(Par1: Par1Type; Par2: Par2Type; ...): ReturnType; stdcall; external 'DLLNAME.DLL' name 'FunctionName' index FuncIndex; // или (если не функция, а процедура): procedure ProcedureName(Par1: Par1Type; Par2: Par2Type; ...); stdcall; external 'DLLNAME.DLL' name 'ProcedureName' index ProcIndex;

Здесь:

• FunctionName (либо ProcedureName) — имя функции (или процедуры), которое будет использоваться в программе.

• Par1, Par2, ... — имена параметров функции или процедуры.

• Par1Type, Par2Type, ... — типы параметров функции или процедуры.

• ReturnType — тип возвращаемого значения (только для функции).

• stdcall — директива, которая должна точно совпадать с используемой в самой DLL.

• external 'DLLNAME.DLL' — директива, указывающая имя внешней DLL, из которой будет импортирована данная функция или процедура.

• name 'FunctionName' ('ProcedureName') — директива, указывающая точное имя функции в самой DLL. Это необязательная директива, которая позволяет использовать в программе функцию, имеющую название, отличное от того которое она имеет в библиотеке.

• index FunctionIndex (ProcedureIndex) — директива, указывающая порядковый номер функции или процедуры в DLL. Это также необязательная директива.

1. Динамическая загрузка DLL. Это гораздо более сложный, но и более элегантный метод. Он лишен недостатка первого метода. Недостаток — объем кода, необходимого для осуществления этого приема, причем сложность в том, что функция, импортируемая из DLL доступна лишь тогда, когда эта DLL загружена и находится в памяти.

Краткое описание используемых этим методом функций WinAPI:

• LoadLibrary(LibFileName: PChar) — загрузка указанной библиотеки LibFileName в память. При успешном завершении функция возвращает дескриптор (THandle) DLL в памяти.

• GetProcAddress(Module: THandle; ProcName: PChar) — считывает адрес экспортированной библиотечной функции. При успешном завершении функция возвращает дескриптор (TFarProc) функции в загруженной DLL.

• FreeLibrary(LibModule: THandle) — делает недействительным LibModule и освобождает связанную с ним память. Следует заметить, что после вызова этой процедуры функции данной библиотеки больше недоступны.

В Java динамическая загрузка классов встроена непосредственно в Java и реализуется классом Class, посредством следующих статических методов: Class.forName() и Class.newInstance().

Вот его формальное объявление:

public static Class forName(String className)

Throws ClassNotFoundException

Метод отыскивает в системе (среди путей поиска классов CLASSPATH) класс с заданным именем className и возвращает соответствующий экземпляр класса. Имя класса должно быть полным, то есть включать имя пакета. Если такой класс отсутствует, возбуждается исключение ClassNotFoundException. После получения переменной типа Class, следующее наиболее типичное действие — создание экземпляра только что загруженного класса. Для этого служит метод Class.newInstance(). Его объявление:

public Object newInstance()

throws InstantiationException, IllegalAccessException

Чтобы этот метод мог выполниться, у загруженного класса должен существовать пустой конструктор (без аргументов), либо — что по существу то же самое — не должно быть описано вообще никаких конструкторов. В противном случае будет возбуждено исключение InstantiationException.

Нетрудно заметить, что в Java реализовать динамическую загрузку классов по заданному имени наиболее удобно, чем и обуславливается выбор средств реализации программной оболочки поддержки и синтеза рациональных решений.

2. ФУНКЦИОНАЛЬНЫЕ БЛОКИ ПРОГРАММЫ

В состав программы входят следующие функциональные блоки:

1) блок организации работы приложения;

2) блок визуализации интерфейса программы;

3) блок справочной информации;

4) блок вспомогательных модулей;

5) блок визуализации методов принятия и синтеза рациональных решений;

6) блок реализации алгоритмов принятия и синтеза рациональных решений;

7) блок вывода результатов.

На рис. 1. представлена схема взаимодействия функциональных блоков с указанием модулей, входящих в каждый блок.

Блок организации работы приложения отвечает за реализацию организации работы приложения. Блок визуализации интерфейса программы обеспечивает визуальное восприятие программной оболочки. Блок справочной информации отвечает за создание окна отображения справочной информации в HTML браузере. Блок вспомогательных модулей — это блок, содержащий модули компоновки элементов пользовательского интерфейса, контроля расширений файлов, создания списка из элементов CheckBox. Блок визуализации методов принятия и синтеза рациональных решений отвечает за визуализацию реализованных методов. Блок реализации алгоритмов принятия и синтеза рациональных решений реализует непосредственно модули реализации методов принятия и синтеза рациональных решений. Блок вывода результатов отвечает за создание отчетов по результатам работы.

3. ЭРГОНОМИЧЕСКИЙ АСПЕКТ СОЗДАНИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Проектирование пользовательского интерфейса чаще всего происходит неосмысленно, без учета эргономических требований. Разработанная программная оболочка удовлетворяет эргономическим показателям [3, 4] для наилучшего восприятия информации пользователем.

Эргономика предъявляет определенные требования к взаимодействию человека и машины. Основными объектами выступают объем предоставляемых сведений, расположение объектов, их очередность, цветовое оформление.

В процессе разработки программы были учтены основные эргономические показатели, для улучшения восприятия информации. Так, например, в матрице попарных сравнений числовые данные (количественные оценки) выровнены справа до десятичной запятой.

Кроме того, были использованы следующие виды кодирования:

- Кодирование формой.

Узлы иерархии в методе МАИ представлены в виде замкнутой фигуры. Так как фигуры, контур которых составлен из прямых линий, различаются легче, чем имеющие кривизну и много углов. Хорошо зрительно воспринимаются фигуры, которые имеют выступы. Поэтому предпочтительнее изображать узлы иерархии в виде обычного прямоугольника или же прямоугольника со скругленным концами.

При построении контуров фигур не используются пунктирные и штриховые линии, которые затрудняют восприятие информации.

- Кодирование размером.

Более важная информация имеет размер больше, чем остальная.

- Буквенно-цифровое кодирование.

Чтобы исключить смешение информации, основные параметры: высоты, толщины, ширины линии применены ко всему оформлению программы. Для числовых значений используются только арабские цифры.

- Кодирование цветом.

Оптимальными цветами являются красный, зеленый, красный, голубой, желтый, фиолетовый. Пользователь может сам выбрать наиболее удобную для него восприятия цветовую гамму, при помощи редактора проекта (рис. 2).

Здесь недопустимы сочетания близких цветов, например, красные символы на розовом фоне, голубые на зеленом, желтые на белом, черные на синем, и наоборот. В качестве фона можно использовать различные текстуры, в том числе и с изменяемым рисунком. Вот краткий перечень распространенных цветовых сочетаний в порядке постепенного ухудшения их восприятия человеком:

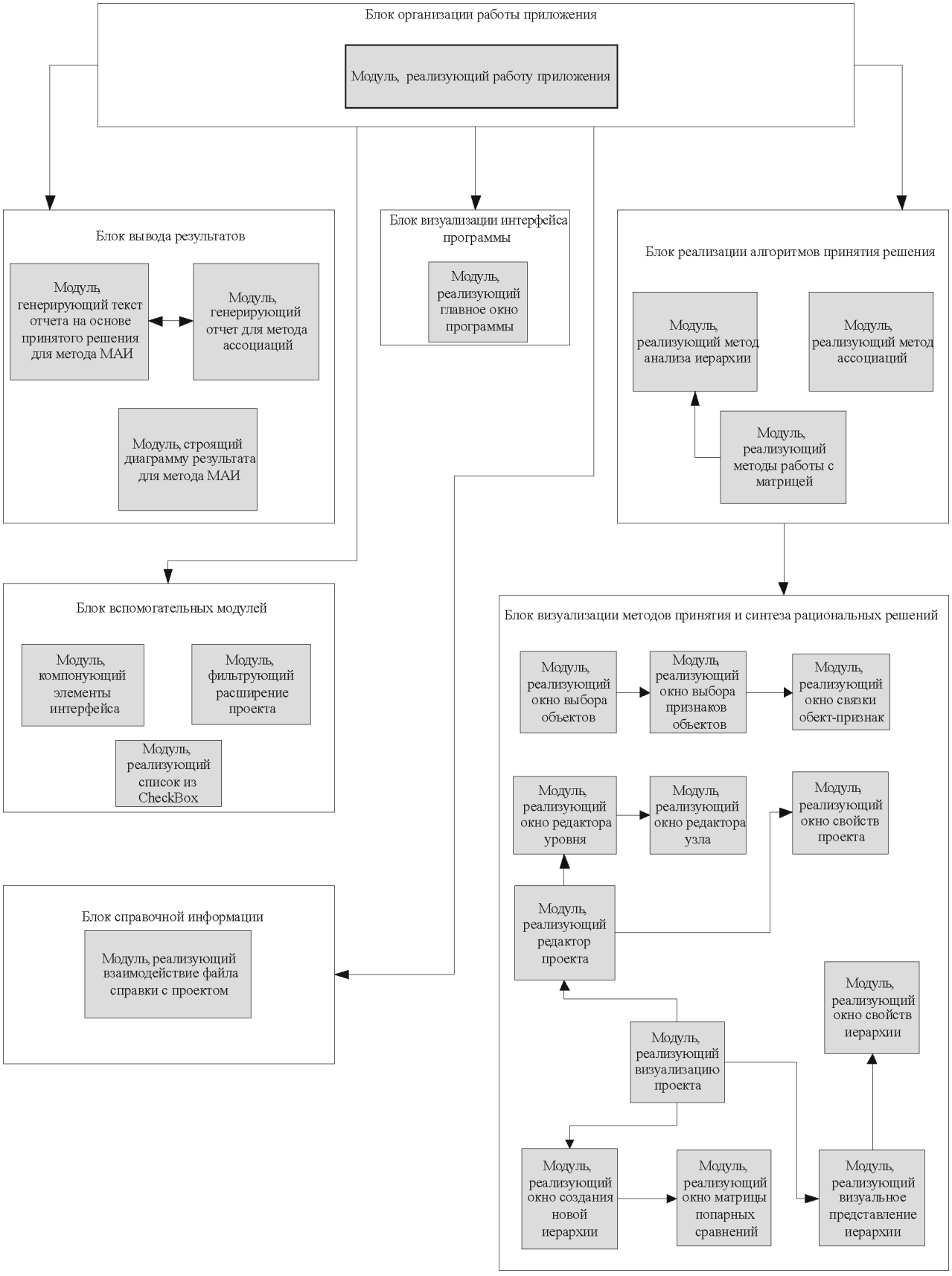

Рис. 1. Взаимодействие функциональных блоков

— синий на белом;
— черный на желтом;
— зеленый на белом;
— черный на белом;
— зеленый на красном;
— красный на желтом;
— красный на белом;
— оранжевый на черном;
— черный на пурпурном;
— оранжевый на белом;
— красный на зеленом.

По умолчанию (рис. 3) предложено следующее цветовое решение: цвет узлов — желтый, цвет текста — черный, наиболее значимая информация выделена жирным шрифтом.

ЗАКЛЮЧЕНИЕ

Реализованная программная оболочка, предназначенная для поддержки принятия и синтеза рациональных решений, позволяет пополнять набор методов поддержки принятия решений при помощи подключения модулей реализаций через внешний интерфейс программы. Пользователь, расширяя функционал программы, может адаптировать систему под довольно специфические задачи. Такой подход дает право использовать программу, не ограничиваясь возможностями, заложенными при разработке. Явным преимуществом предложенного подхода является также и то, что реализация новых методов осуществляется независимо

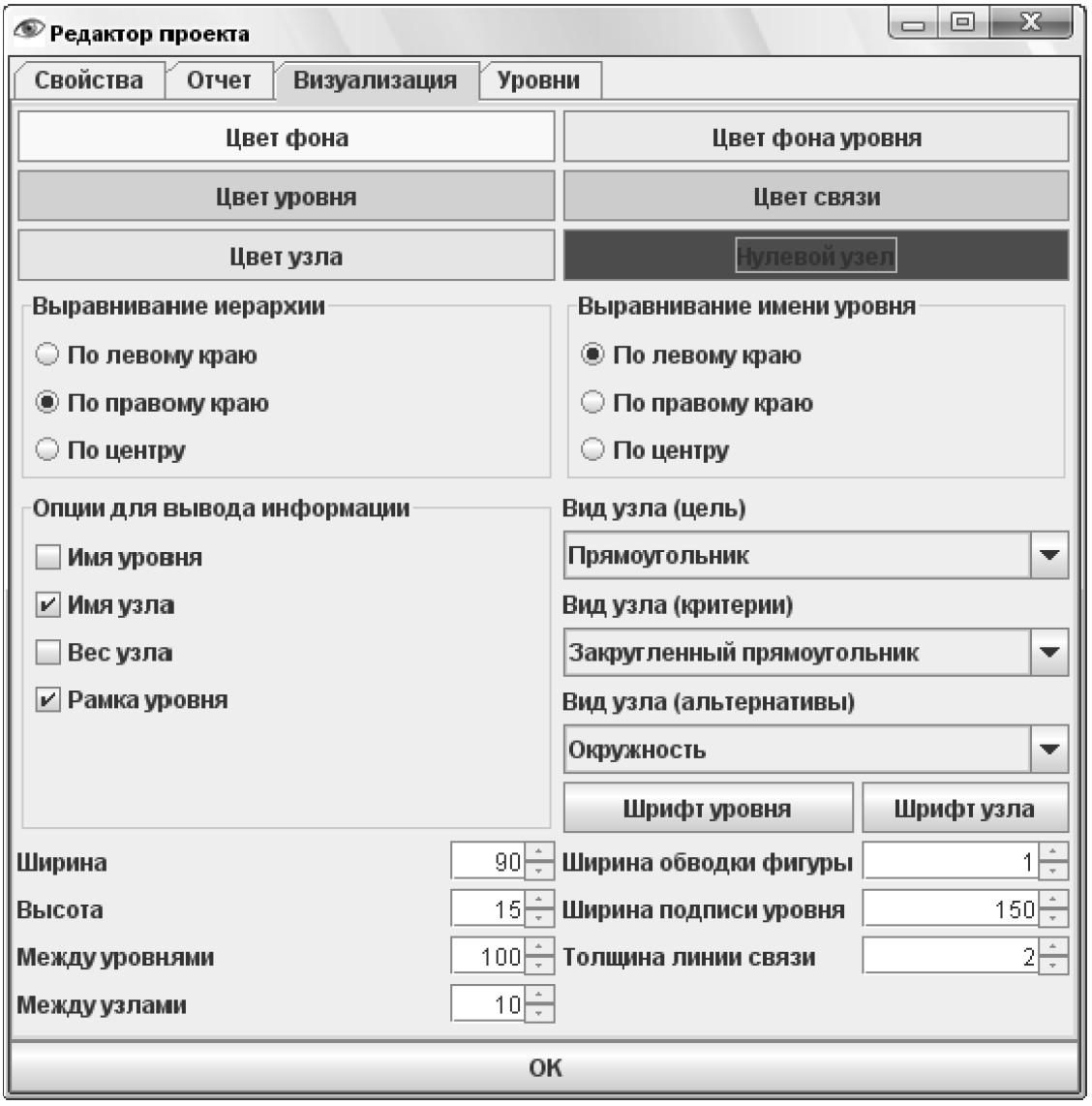

Рис. 2. Редактор проекта

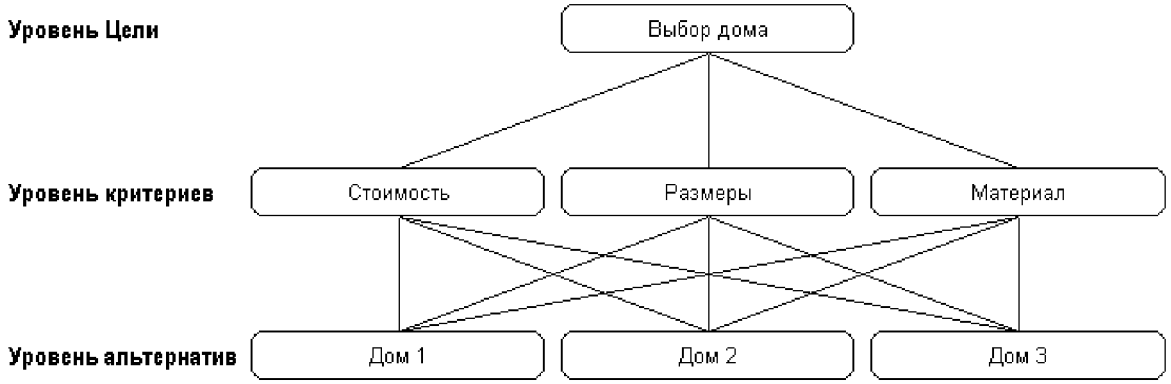

Рис. 3. Визуализация иерархии

друг от друга и добавление новых функций не требует повторного тестирования уже существующих.

Возможности программной оболочки проиллюстрированы работой двух методов: метода анализа иерархий (МАИ) и метода ассоциаций. Выбор методов обусловлен тем, что МАИ является одним из наиболее распространенных и интересен решением визуального представления иерархии при многокритериальном выборе. В методе МАИ процесс постановки задачи достаточно неоднозначен, часто сложен. Задача задается в виде иерархии, с различными уровнями (глобальная цель, критерии, альтернативы решения поставленной задачи). В свою очередь, критерии могут иметь подкритерии, тем самым, увеличивая число парных сравнений критериев (подкритериев, альтернатив), что делает процесс принятия решения весьма трудоемким. При решении небольших задач достаточно адаптированного интерфейса пользователя, тогда работа будет сведена к оценке попарно ряда вариантов (последовательный обход ветвей иерархии), оформленных в виде списка вопросов. В случае же более сложной задачи, актуальна визуализация иерархии со всеми критериями, подкритериями и альтернативами, с возможностью интерактивного редактирования задачи в процессе решения.

Реализованный метод ассоциаций генерирует гирлянду вида «признак» — «объект», что позволяет использовать подобный метод при поиске новых модификаций известных устройств и способов их использования. Метод ассоциаций — это один из методов синтеза новых решений.

В ходе разработки были учтены основные эргономические требования к программному интерфейсу. Эргономичные системы работают именно так, как пользователи ожидают, и позволяют пользователям фокусироваться на собственных задачах, а не особенностях взаимодействия с системой. Системы, удовлетворяющие эргономическим параметрам, проще изучить, они более эффективны, они также позволяют минимизировать количество человеческих ошибок и увеличить субъективную удовлетворенность пользователей.

Важной особенностью программной оболочки является то, что ее можно применять в учебной деятельности (в частности, в дистанционном образовании и в качестве инструмента изучения материала). Пользователь может не просто изучить сами методы и алгоритмы, но и протестировать их на конкретных задачах в различных областях.

Кроме того, программа может являться инструментом решения задач исследовательского характера (в том числе и в учебном процессе). Дело в том что, при проведении исследований изучение самого метода, его математического аппарата не всегда необходимо. В некоторых случаях достаточно лишь четко сформулировать задачу, ввести начальные данные (экспертную информацию), и необходимый результат будет получен без лишних временных затрат. Такая ситуация часто имеет место при проведении исследований в гуманитарных областях.

Таким образом, программная оболочка, позволяет:

• подключать модули реализаций различных методов анализа и синтеза рациональных решений через внешний интерфейс программы;

• решать задачи методом МАИ;

• решать задачи методом ассоциаций.

Оболочка разработана с учетом эргономических требований, предъявляемых к программному интерфейсу.

Программное средство «Программная оболочка поддержки и синтеза рациональных решений» зарегистрировано в Государственном информационном фонде неопубликованных документов ФГНУ «Центр информационных технологий и систем органов исполнительной власти» (№ 50200900857 от 20.07.2009).

СПИСОК ЛИТЕРАТУРЫ

1. *Леденева Т. М.* Обработка нечеткой информации: учебное пособие / Т. М. Леденева. Воронеж: Воронежский государственный университет, 2006. — 233 с.

2. *Алгазинов Э. К., Сирота А. А.* Анализ и компьютерное моделирование информационных процессов и систем / Э. К. Алгазинов, А. А. Сирота — М.: Диалог-МИФИ, 2009. — 416 с.

3. *Андреев В.* Памятка по эргономике для разработчиков / В. Андреев. — (http://www.usability.ru/Articles/instruction.htm).

4. *Мандел Т.* Разработка пользовательского интерфейса / Т. Мандел. Пер. с англ. — М.: ДМК Пресс, 2001. — 416 с.

Воронина Ирина Евгеньевна — к.т.н., доцент кафедры ПОиАИС, факультет прикладной математики информатики и механики, Воронежский государственный университет. Тел. 208-337. E-mail: voronina@amm.vsu.ru

Митрофанова Елена Юрьевна — ассистент кафедры информационных систем, факультет компьютерных наук, Воронежский государственный университет. E-mail: mitrofanova_lena@bk.ru

Voronina I. Ye. — Candidate of Technical Sciences, Associate Professor, the dept. of Software and Information System Administration, Voronezh State University. Tel. 208-337. E-mail: voronina@amm.vsu.ru

Mitrofanova E.Yu. — Assistant, the dept. of Information Systems, Voronezh State University. E-mail: mitrofanova_lena@bk.ru