УДК 681.3

СИСТЕМА ПОДДЕРЖКИ ПОЛЬЗОВАТЕЛЯ НА ОСНОВЕ ЭКСПЕРТНОЙ СИСТЕМЫ

А.С. Пригожев, Е.И. Алёшкина, М.В. Бабичева, А.В. Вовк, В.А. Писаренко

Одесский национальный политехнический университет,
65044, Одесса, проспект Шевченко, 1.
Тел.: 8 (048) 779 7566; 8 (048) 779 7106,
prigozhev@matrix.odessa.ua

Рассматриваются подходы к проектированию экспертной системы поддержки пользователя. Сформулированы основные теоретические положения, на которых основывается разработка базы знаний для экспертной системы. Разработаны методы сбора статистики о действиях пользователя. Разработана структура экспертной системы поддержки пользователя.

In article approaches to designing expert system of support of the user are considered. The basic theoretical positions on which development of the knowledge base for expert system is based are formulated. Methods of gathering of statistics about actions of the user are developed. The structure of expert system of support of the user is developed.

Введение

Широкое внедрение систем поддержки пользователя привело к необходимости разработки специализированных систем поддержки пользователя. Современные средства поддержки пользователя основаны на применении различных текстовых руководств а также дружественных мастеров [1]. С развитием аппаратно-программных средств стало возможным использование для решения указанной задачи средств искусственного интеллекта. Это привело к появлению различных адаптивных интерфейсов, которые позволяли настроить интерфейс на индивидуальные особенности восприятия информации конкретным пользователем. Однако, несмотря на использование таких средств, остаётся задача поддержки пользователя в процессе решения задачи с заранее заданными параметрами. Одним из путей решения данной задачи является построение экспертных систем для поддержки пользователя. Использование таких систем позволит гибко настраивать систему поддержки под конкретную решаемую пользователем задачу. В качестве исходных данных для системы приобретения знаний, входящей в состав экспертной системы, предлагается использовать интерфейсы пользователя и алгоритмы решения пользовательских задач.

Описание взаимодействия пользователя и АИС

Для описания взаимодействия пользователя и АИС предложено использовать абстрактную схему управления, предложенную В.М. Глушковым [2, 3].

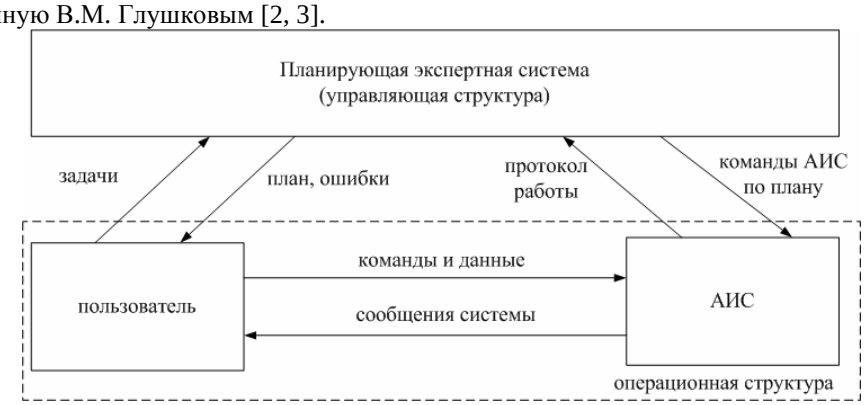

Рис. 1. Схема управления взаимодействием «пользователь-АИС»

Управляющая структура должна содержать сведения о предметной области АИС и о пользователе. Для описания модели пользователя предлагается выбрать автоматную модель. В качестве средства описания задач в заданной предметной области применим модель представления знаний в виде сценариев [2, 3]. Тогда представленную на рис. 1 модель можно представить как показано на рис. 2.

Синтез базы знаний по алгоритмам решения задач

В работах [4, 5] предложено строить базу знаний для экспертной системы на основе дерева задач. Рассмотрим далее способы получения базы.

При решении задачи поддержки пользователя необходимо знать, при каких условиях команда является доступной для пользователя и когда можно завершить циклическое выполнение команды.

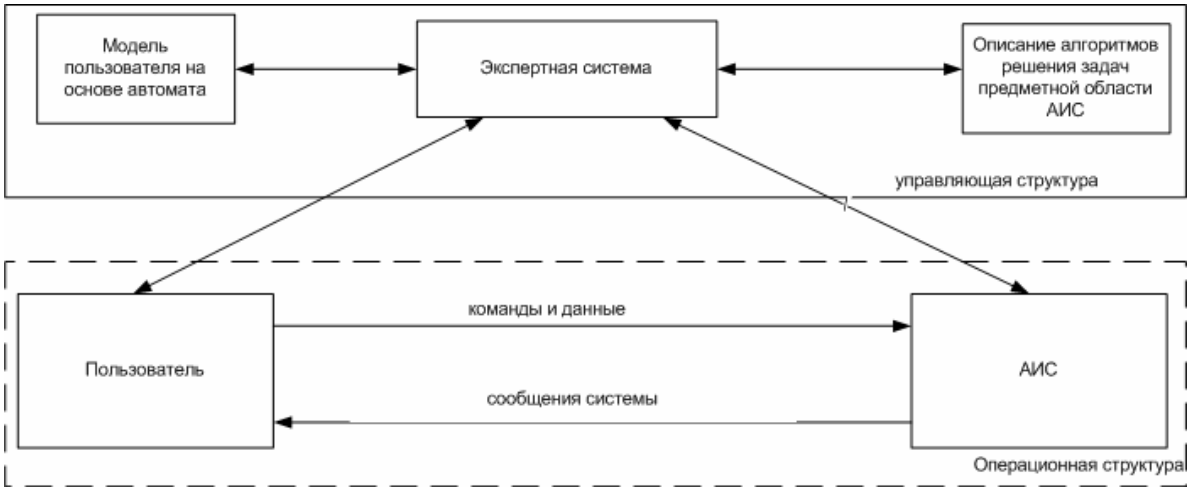

Рис. 2. Детализированная абстрактная схема управления «пользователь-АИС»

Для принятия решения о том, является ли данная команда допустимой для выполнения в определённый момент времени, предлагается выполнение каждой команды представлять в виде альтернативы вида:

$$SUBP ::= ([u_a]A, E). \tag{1}$$

Если команда включена в алгоритм решения задачи и может быть выполнена, то при выполнении условия альтернативы, сопоставленного команде, она должна быть выполнена пользователем хотя бы один раз. Некоторые команды для успешного решения задачи должны быть выполнены несколько раз. Выполнение команды может быть прервано по необходимости либо в силу недоступности команды для пользователя. В общем виде представление циклично выполняемой команды в виде структурной схемы алгебры Дейкстры может быть представлено следующим образом:

$$SUBP' ::= \{A[u]\}. \tag{2}$$

Подставляя схему (1) в схему (2), а также выделяя условие альтернативы в отдельное условие, входящее в условие цикла, построена алгебра сценариев, которая представляет собой алгебру структурных схем. Общий вид данной алгебры представлен далее [6, 7]:

$$ALGS ::= <\{([u_a]\{A[\overline{u}_a \vee u_i]\}; E); \{A * B\}; L(2)\}; \{SUPER\}> \tag{3}$$

В качестве второй основы в данной алгебре используется операция композиции. На основании предложенной алгебры предложено строить иерархию задач, используя граф. Рассмотрим математическое описание данного графа.

Для построения графа выделим наиболее чётко формализуемые признаки АИС. К ним относятся: команды, алгоритмы решения задач, структура задач. Под структурой задач понимается взаимосвязь задач и подзадач.

Исходные данные для построения модели представления знаний для системы «пользователь-АИС» – это множество задач T, решаемых в АИС. Среди существенных свойств задач выделим: название и описание задачи, исходные данные, команды для решения задачи, алгоритм её решения и описание результатов. Под T^2 понимается декартово произведение, которое определяет множество всех пар задач:

$$T^2 = T \times T = \{(t_i, t_j) \in T^2 : t_i, t_j \in T\}. \tag{4}$$

Определим теперь формально отношение «быть подзадачей» на множестве задач.

Отношением «быть подзадачей» назовём множество пар идентификаторов $(t_i, t_j) \in T^2$ таких, что для задачи t_j выполняется определение подзадачи задачи t_i.

Обозначив введённое отношение St , можно записать:

$$S_t = \{(t_i, t_j) \in T^2 | t_j \ \textit{подзадача} \ t_i\} \tag{5}$$

Отношение (5) иллюстрирует связь задач и подзадач АИС. Следовательно, граф этого отношения можно использовать для моделирования ситуаций в процессе взаимодействия [7]. Рассмотрим особенности графа отношения S_t. Задачи, возникающие перед пользователем в процессе его работы в АИС, являются алгоритмически

разрешимыми. Поэтому под включением процедуры решения подзадачи в процедуру решения задачи понимается строгое включение структурной схемы алгоритма решения подзадачи в структурную схему алгоритма решения задачи. Следовательно, отношение S_t иррефлексивно, поэтому граф отношения S_t не содержит петель. В силу определения понятия подзадачи, отношение S_t транзитивно. Отношение S_t является несимметричным, в силу определения понятия подзадачи. Это означает, что граф указанного отношения будет содержать только направленные рёбра.

В силу транзитивности рассматриваемого отношения, для облегчения вывода в процессе работы экспертной системы, необходимо устранить из рассмотрения транзитивные рёбра. Поэтому введём в рассмотрение множество $T' \subset T$ подзадач только одной задачи. Определим такое множество формально:

$$T' = \{t_i : (t_0, t_i) \in S_t\}. \quad (6)$$

В формуле (6) t_0 – задача, не являющейся подзадачей ни одной задачи из множества T'. Определим множество $S_t' \subset S_t$:

$$S_t' = \{(t_i, t_j) : (t_i, t_j) \in S_t, t_i \in T', t_j \in T'\}. \quad (7)$$

Построенное отношение будет содержать транзитивные пары, а на графе отношения будут присутствовать транзитивные рёбра. Далее будем рассматривать поддерево непосредственных подзадач, которое является остовным деревом для графа задач только одной задачи. Определим такое дерево формально, введя следующее определение.

Определение. t_j является непосредственной подзадачей задачи t_i, если:

1. $(t_i, t_j) \in S_t'$.

2. Не существует такого $t_k \in T$, что $(t_i, t_k) \in S_t'$ и $(t_k, t_j) \in S_t'$.

Исходя из приведенного определения непосредственной подзадачи и определения остова [8], можно утверждать, что остов графа задач содержит только непосредственные подзадачи. Корнем дерева является задача, не являющаяся подзадачей ни одной задачи, входящей в дерево. Формулировка этой задачи совпадает с целью диалога пользователя и АИС. Формально данное условие можно записать таким образом:

$$\forall t_i \in T', (t_i, t_0) \notin S_t' \quad (8)$$

На основе вышеприведенного определения введём множество S_t^0, элементами которого являются пары элементов множества T', удовлетворяющие определению непосредственной подзадачи. Формально это можно записать следующим образом:

$$S_t^0 = \{(t_i, t_j) \in T'^2 | t_j \text{ непосредственная подзадача } t_i\}. \quad (9)$$

Результаты решения задачи могут зависеть от порядка решения подзадач. Это связано с тем, что в общем случае операция композиции не коммутативна. Поэтому для определения порядка выполнения подзадач, являющихся непосредственными для одной задачи, каждой задаче поставим в соответствие номер, и тогда предполагаем, что подзадачи одного уровня решаются в порядке возрастания номеров. Отметим также, что в случае, если известно заранее, что перестановка действий не влияет на окончательный результат работы программы, номера задач могут совпадать.

Основой для рассматриваемого условного графа задач является множество S_t^0. Результат решения любой задачи может быть сформулирован в виде выражения из логико-функциональной модели процесса решения задачи. Логико-функциональная модель задаёт некоторое множество условий U. Введём в рассмотрение два отображения: S_{tg} , определяющее условие альтернативы, и A_0 , определяющее условие цикла.

$$S_{tg} : S_t^0 \to U. \quad (10)$$

$$A_0 : S_t^0 \to U. \quad (11)$$

Сопоставим каждой вершине графа команду для решения некоторой задачи. Если вершина графа содержит подзадачи, то ей сопоставляется специальная команда «алгоритм», которая говорит о том, что данную задачу можно решить с помощью алгоритма, состоящего в определённой последовательности команд АИС. Предлагаемое расширенное множество команд АИС будем обозначать буквой $C^{\#}$. Определим отображение A:

$$A : T' \to C^{\#}. \quad (12)$$

Таким образом, нагруженный граф задач можно охарактеризовать подмножеством B декартова произведения:

$$B \subset T^{'} \times T^{'} \times U \times U \times C^{\#} \,. \tag{13}$$

Определим это множество формально:

$$\begin{aligned} \mathrm{B} = \{(\mathrm{t_i}, \mathrm{t_j}, \varphi_1, \varphi_2, \mathrm{c}) : (\mathrm{t_i}, \mathrm{t_j}) \in \mathrm{S_t^0}, \varphi_1 = \mathrm{S_{tg}}(\mathrm{t_i}, \mathrm{t_j}), \\ \varphi_2 = \mathrm{A_0}(\mathrm{t_i}, \mathrm{t_j}), \mathrm{c} = \mathrm{A}(\mathrm{t_j})\} \end{aligned} \tag{14}$$

Пусть B_i – это множество пятёрок из B, у которых первая компонента равна t_i. Тогда B_i образует сценарий для решения задачи.

Рассматриваемое множество B_i представляет собой описание формального графа задач. Рассмотрим базовые элементы такого графа, которые отображают основные компоненты алгебры сценариев.

Для представления структурной схемы (1) используется следующее ребро графа, показанное на рис. 3.

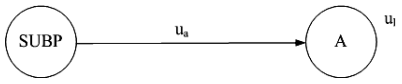

Рис. 3. Представление схемы SUBP на дереве задач

Операция композиции на дереве задач представляется в виде, показанном на рис. 4.

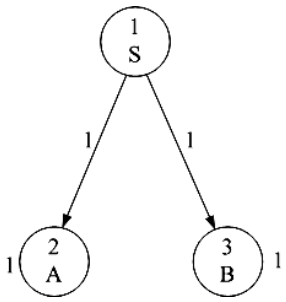

Рис. 4. Представление композиции на графе задач

Предложена информационная технология позволяющая автоматизировать процесс синтеза дерева задач и информационная технология, позволяющая получить базу знаний (рис. 5).

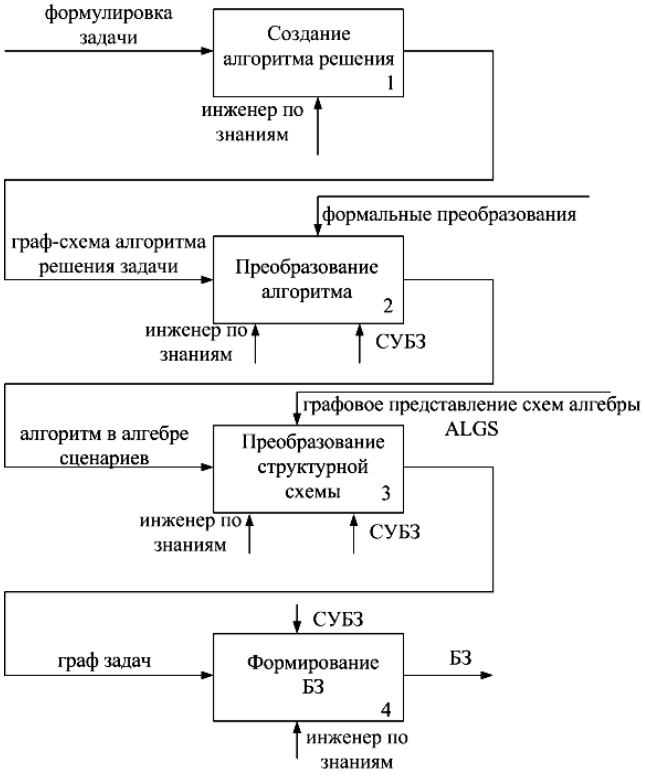

Рис. 5. Информационная технология

Однако представление базы знаний в виде дерева задач является достаточно избыточным в силу присутствия в разных деревьях одинаковых идентификаторов задач. Учитывая данный недостаток предложено усовершенствовать представление знаний в виде дерева задач объеденив в единый граф деревья, имеющие общие задачи. Также, для обеспечения механизма обратной связи с пользователем, позволяющей усовершенствовать в дальнейшем пользовательский интерфейс программного обеспечения, предлагается сопоставить каждому графу ряд характеристик, которые будут описаны далее.

Синтез базы знаний с использованием пользовательского интерфейса

При анализе интерфейса пользователя с целью построения базы знаний может применяться один из следующих способов:

1. Анализ формального описания формы в некоторой среде программирования.
2. Анализ взаимодействия пользователя и системы в on-line или off-line режиме.

При построении формы в любой среде разработки создаётся файл формы, в котором указывается вся информация о форме и её элементах, такая как тип и название элемента, его содержимое, размеры, положение на форме, используемый шрифт и т.д. Рассмотрим структуру файла формы на примере .dfm-файла, создаваемого средой разработки Delphi.

Файл формы (.dfm) – файл, содержащий сведения об опубликованных (т. е. доступных в инспекторе объектов) свойствах компонентов, содержащихся на форме. Двоичный файл формы содержит информацию, используемую для конструирования формы из компонентов, расположенных на ней. При добавлении компонента к форме двоичный файл формы модифицируется. При редактировании свойств компонента в инспекторе объектов эти изменения сохраняются в файле формы.

Отметим, что при изъятии какого-либо компонента из формы в буфер обмена, в последнем реально оказывается часть тестового представления файла формы, содержащая описание данного компонента. В этом можно убедиться, выполнив затем операцию вставки из буфера обмена в любом текстовом редакторе.

Русский текст в заголовках, названиях, содержимом элементов отображается в виде ASCII-кода побуквенно, английский текст отображается в нормальном виде. Существует иерархия вложения описания элементов. Корневым элементом можно назвать форму. Далее идут элементы, расположенные непосредственно на форме, далее элементы, расположенные внутри других элементов (например RadioButton внутри GroupBox). Для всей формы указываются вначале шрифт, цвет шрифта, его размер и т.д., а так же дополнительные свойства, например, какой будет форма при старте программы – свернутой или развернутой. Если свойства элемента отличаются в чем-то от свойств формы, например, используется другой шрифт, то параметр ParentFont (для случая со шрифтом) этого элемента устанавливается в False и указываются новые значения отличающихся свойств. Если к какому-либо событию элемента привязана процедура или функция, то указывается событие и соответствующее ему название процедуры или функции, описанной в .pas файле. Также, некоторые значения параметров по умолчанию не указываются, например, для CheckBox - Checked = False, State = cbChecked, однако, если изменить значения этих параметров, они отобразятся с новыми значениями в .dfm-файле.

Каждый элемент формы, как и сама форма, представлены отдельным объектом.

```
object Edit1: TEdit
```

Описание объекта включает в себя название и тип, идущие через двоеточие. Все свойства объекта и описания вложенных в него объектов находятся между ключевыми словами begin и end. Описание свойства или параметра представляет собой название и значение разделенные знаком «=». Значения, представляющие собой текст, заключаются в одинарные кавычки.

```
Left = 40
Caption = 'Path to file:'
```

Если значение представляет собой совокупность строк (например, в случае с ListBox), то оно дополнительно заключается в круглые скобки и указывается построчно.

```
Items.Strings = (
                    'String 1'
                    'String 2'
                     'String 3')
```

Парсер .dfm – файла представляет собой программу, разбивающую файл формы на отдельные слова . Парсер игнорирует такие символы как пробел, переход на следующую строку, символ равенства и некоторые другие, тем самым позволяя получать отдельные слова без лишних символов. Исключение составляют слова, заключенные в одинарные кавычки. Такую конструкцию парсер рассматривает как текст и работает с ней, как с

одним словом. Полученные после разбиения слова сравниваются с некоторыми заранее заданными текстовыми константами. Совпадение с той или иной константой влечет за собой запись конкретного свойства в таблицу объектов. Таблица объектов включает в себя следующие поля:

1. Тип.
2. Название.
3. Надпись.
4. Текст.
5. Состояние (отмечено/не отмечено).
6. Порядок перехода по Tab.
7. Нахождение в группе.
8. Идентификационный номер.

Таким образом, после статического анализа файла формы имеем возможность получить информацию об элементах формы и их основных свойствах. Кроме того оригинальные текстовые идентификаторы, сохраняющиеся во втором поле таблицы не всегда удобны для оперирования, в отличие от числовых идентификаторов (восьмое поле таблицы), присваиваемых парсером сквозной нумерацией всем встреченным элементам. Таблица дает возможность сразу получать всю необходимую информацию об объекте, при его активации во время динамического анализа.

Однако описание файла формы доступно для анализа далеко не всегда, и поэтому возникает необходимость в динамическом анализе формы, непосредственно в процессе работы пользователя. Поэтому необходимо создавать лог-файл, в котором отображается весь процесс взаимодействия пользователя с системой.

Каждый элемент интерфейса имеет свой идентификационный код или просто идентификатор, присвоенный ему при создании. Такой код можно получить при активации элемента, например курсором мыши. Также в процессе взаимодействия

При работе пользователя с формой формируется цепочка идентификаторов элементов, с которыми он работал (т. е. активировал тем или иным образом) – статистика (или так называемый лог) работы пользователя. Туда заносятся подряд все идентификаторы элементов в порядке, в котором с ними работал пользователь.

Для простоты предположим, что каждая логически завершенная последовательность действий пользователя оканчивается нажатием на кнопку (например, «ОК» или «Принять»). В таком случае, идентификаторы кнопок можно перенести в отдельную категорию неких контрольных точек (КТ). Они являются разделителями, разбивающими весь лог на решение отдельных подзадач (например, заполнить регистрационную форму и подтвердить заполнение кнопкой «ОК»). КТ всегда стоят в конце некоторой последовательности идентификаторов, по сути являющейся решением подзадачи.

Нажатие кнопки «Отмена» либо эквивалентной ей, так же как и закрытие окна приводит к удалению из лога всех идентификаторов вплоть до первой встретившейся КТ, так как такая последовательность считается незавершенной или неверной.

Если в решении (цепочка идентификаторов, завершающаяся КТ) найдены два или более одинаковых идентификатора, то можно считать это ошибкой пользователя и удалить все одинаковые идентификаторы, кроме одного. Однако, если в окне нет GroupBox или если нажатия RadioButton происходят в рамках одного GroupBox, тогда удаляем все идентификаторы, относящиеся к элементам RadioButton, относящихся к одному GroupBox или окну, кроме последнего, правильного нажатия. В случае с CheckBox необходимо определить четность количества одинаковых идентификаторов. Если количество чётно – то удаляем все одинаковые ID, если нечётно – оставляем один. Таким образом, мы определяем, осталось ли состояние CheckBox неизменным или изменилось в результате действий пользователя.

Далее, разбиваем непрерывную цепочку идентификаторов (лог) по КТ (выделяем решения), причём каждая цепочка начинается с любого идентификатора кроме КТ, а заканчивается обязательно КТ. Затем упорядочиваем идентификаторы, идущие до КТ по возрастанию (либо по убыванию, что несущественно), при этом КТ не подлежит сортировке и остаётся в конце решения. Это позволяет выявить одинаковые решения, даже если порядок нажатий на элементы у разных пользователей был разным, и удалить дубли какого-либо решения.

Предполагая, что известно, какая цепочка, какую задачу (подзадачу) решает, можно отобрать оптимальное решение для экономии усилий пользователя. Для этого при отборе оптимального решения должны учитываться такие параметры как TabOrder элементов в окне, их взаимное расположение и количество элементов, задействованное в данном решении (длина цепочки).

В результате, в базовом варианте можно представить базу знаний (БЗ) в виде таблицы.

Характеристики для определения степени решения задачи

Введём ряд дополнительных характеристик, позволяющих получить некоторую информацию о работе пользователя в системе и контролировать решение в процессе задачи. Для определения характеристики «степень решения задачи» определим понятие «элементарной задачи», как задачи, не имеющей подзадач в контек-

сте рассматриваемой предметной области. Характеристика «степень решения задачи» определяется тогда следующим образом

$$s(t) = \begin{cases} 0, \text{если } t - \text{не решённая элементарная задача} \\ 1, \text{если } t - \text{решённая элементарная задача} \\ \dfrac{\sum_{i=1}^{N} s_i(t)}{N} - \text{если } t - \text{задача, содержащая } N \text{ подзадач} \end{cases} \quad (15)$$

Диапазон значения степени решения задачи возрастает от 0 до 1. Значение 0 – это не решенная задача, 1 – это полностью решенная задача, а значение из промежутка [0;1] – частично решенная задача.

В ходе решения задач в экспертной системе пользователь просматривает иерархию задач и подзадач и отмечает метками решённые, частично решённые и нерешённые задачи. Под глубиной решения задачи будем понимать максимальную глубину отметки в поддереве, соответствующем некоторой задачи. Параметры глубина и степень решения задачи позволяют контролировать, насколько хорошо пользователь знаком с системой.

Приведенные параметры позволяют осуществлять контроль за выполнением задачи пользователем.

Программная реализация системы поддержки

Экспертная система поддержки может работать в автономном режиме, когда пользователь самостоятельно отмечает решённые задачи, отмечая необходимые вершины, а экспертная система проверяет отмеченные вершины. Однако при таком режиме работы существует недостаток – пользователь, недостаточно хорошо знакомый с программной системой, может отметить задачи, которые он не решил. В этом случае существует возможность интеграции разрабатываемой экспертной системы с программной системой в двух режимах – режиме наблюдения за диалогом и режиме автоматизированного выполнения команд.

Основной целью режима наблюдения за диалогом является поддержка пользователя непосредственно в процессе его работы с приложением. Для реализации режима наблюдения за диалогом необходимо решить следующие задачи: перехват сообщений в режиме реального времени от пользователя к программной системе и обратно, а также интерпретация сообщений в терминах экспертной системы поддержки пользователя.

Для решения этих задач необходимо разработать два дополнительных интегрирующих блока: монитор команд и интерпретатор «команда – задача». Монитор команд позволяет решить первую из двух задач. Поскольку решение данной задачи в значительной степени зависит от аппаратно-программного обеспечения АИС, то данный блок и его реализация являются уникальными для данной АИС.

Для решения второй задачи разработаны интерпретатор «команда – задача», который позволяет сопоставить команду АИС некоторому идентификатору задачи в БЗ. При работе данный блок использует таблицу соответствия команд и задач, которая разрабатывается для каждой АИС отдельно, как и БЗ. В ней каждой команде сопоставлена задача, для решения которой используется данная команда.

Рассмотрим функционирование системы в этом режиме. Монитор фиксирует сообщения, передаваемые программной системе, и передаёт их интерпретатору «команда – задача». Интерпретатор «команда – задача» имеет свой внутренний буфер, в который помещается переданная команда, после чего интерпретатор обращается к таблице соответствия задач и команд. Если подцель в БЗ найдена, то в этом случае решатель отмечает данную задачу в рабочей области и очищает свой внутренний буфер команд. Структура системы в нотации UML показана на рис. 6, где каждый функциональный блок системы представлен в виде диаграммы пакетов.

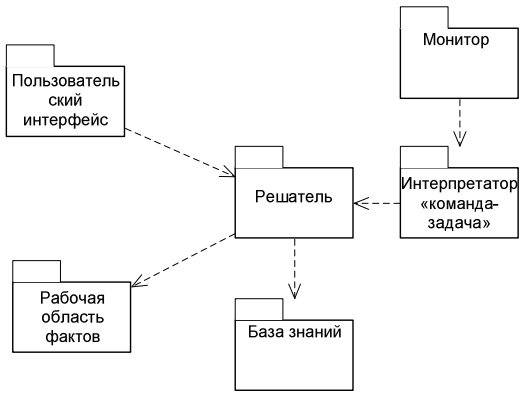

Рис. 6. Структура системы в режиме наблюдения за диалогом

В режиме автоматизированного выполнения команд пользователь имеет возможность выполнить команды системы в автоматизированном режиме. Для реализации этой возможности необходимо решить следующие задачи: интерпретация задачи в виде последовательности команд программной системы и выполнение указанной последовательности с помощью экспертной системы, при этом пользователь вводит данные, а экспертная система передаёт на выполнение в АИС команды.

Для решения указанных двух задач разработаны два дополнительных интегрирующих блока: интерпретатор «задача – команда» и исполнитель команд. Интерпретатор команд получает идентификатор задачи и транслирует его с помощью таблицы соответствия в соответствующую команду, которая далее передаётся эмулятору команд. Общая структура системы в виде пакетов UML представлена на рис. 7.

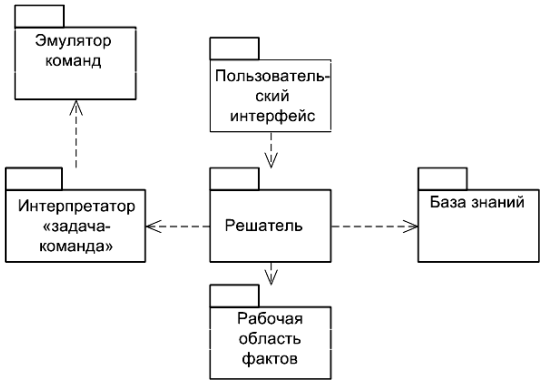

Рис. 7 Структура системы в режиме автоматизированного выполнения команд

Выводы

Описаны подходы к проектированию экспертной системы поддержки пользователя. Предложен математический аппарат, позволяющий реализовать в разработанной системе расширенную функциональность. Описана интеграция экспертной системы с программной системой. Разработанная экспертная система позволяет осуществлять поддержку пользователей непосредственно при решении необходимых им задач. В дальнейшем, при разработке темы, планируется формализовать и разработать структуры данных для реализации автоматной модели поведения пользователя в системе.

1. Денисов Ю.А. Операционные системы: правила работы // URL: http://www.citforum.ru/operating_systems/ois2/index.shtml. (просмотрено 29.05.2007)
2. Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К. Многоуровневое структурное проектирование программ: теоретические основы, инструментарий. – М.: Финансы и статистика, 1989. – 207 с.
3. Глушков В.М. Основы безбумажной информатики. – М.: Наука, 1982. – 552 с.
4. Рувинская В.М., Пригожев А.С. Разработка планирующей экспертной системы для поддержки работы пользователя с программной системой // Вестник ХГТУ. – 2005. – №3. – С. 133–137.
5. Пригожев А.С. Реализация решателя и базы знаний экспертной системы для планирования действий пользователя при разработке программ // Радиоэлектроника и информатика. – 2005. – № 4. – С. 110–116.
6. Пригожев А.С. Информационная технология помощи пользователю // Холодильная техника и технология. – 2007. – № 2. – С. 98-101.
7. Цейтлин Г.Е. Введение в алгоритмику. – К.: Сфера, 1998 . – 310 с.
8. Харари Т. Теория графов. – М.: Мир, 1973 – 300 с.