УДК 28.17.19

ИНТЕРАКТИВНАЯ ИНФОРМАЦИОННАЯ СИСТЕМА С ИНТЕЛЛЕКТУАЛЬНЫМ ЯДРОМ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ С ВЕБ-ИТЕРФЕЙСОМ

© О.В. Крючин, А.А. Арзамасцев, Е.В. Вязовова

Ключевые слова: информационные системы; искусственные нейронные сети; протокол передачи данных.
Описаны принципы функционирования интерактивной информационной системы, ядром которой является аппарат искусственных нейронных сетей. Дано подробное описание интерфейса системы и протокола обмена сообщениями между ее компонентами. Приводятся фрагменты исходного кода модуля взаимодействия ядра и интерфейса системы.

ВВЕДЕНИЕ

В различных предметных областях часто возникают задачи, для решения которых рационально использовать информационные системы (ИС), построенные на основе накопленной базы знаний о рассматриваемой задаче. Формирование такой базы знаний происходит при совместной работе пользователей и эксперта в исследуемой предметной области задачи. При последовательном вводе информации пользователями (входные данные задачи) происходит ее оценка экспертом, при этом определяются соответствующие выходные состояния задачи. Таким образом, происходит накопление базы знаний о рассматриваемой задаче до необходимого объема. Затем происходит построение интеллектуальной модели задачи. Моделирование осуществляется с помощью построения и обучения искусственной нейронной сети (ИНС) на основе накопленных данных [1]. По мере поступления новой информации в базу знаний полученная интеллектуальная модель задачи совершенствуется. При накоплении некоторого критического объема базы знаний построенная модель способна самостоятельно определять решение задачи.

Целью данной работы является разработка целостной интерактивной системы, состоящей из взаимосвязанных компонентов, позволяющих осуществлять построение нейросетевых систем.

ИНФОРМАЦИОННАЯ СИСТЕМА

Общее описание. Пользовательская подсистема ИС состоит из набора *PHP*-скриптов, взаимодействующих с пользователем, и компонента, взаимодействующего с управляющей подсистемой [2].

Подсистема поддерживает три уровня доступа – пользователя, оператора (эксперта в предметной области) и администратора (эксперта в области ИНС). Каждому уровню соответствует определенный набор полномочий и функциональных возможностей. Уровню администратора соответствуют полномочия управления пользователями и настройками ИНС. Уровень оператора наделен возможностями управления объектом. Пользовательский уровень дает возможность вносить информацию о параметрах объекта и получать результат по внесенным данным.

Конкретная задача, рассматриваемая в заданной предметной области, называется в разрабатываемой системе объектом. Объект системы определяется набором совокупностей входных параметров с соответствующими выходными состояниями. Создание объекта доступно для пользователя с полномочиями оператора. В результате создания определяются такие его характеристики, как количество входных параметров, входные параметры, по которым впоследствии будет проводиться анализ, предполагаемые выходные состояния.

Накопление информации об объекте может осуществляться двумя способами.

При первом способе (с участием пользователя и оператора) пользователь, зарегистрированный в системе, вносит параметрические данные объекта, которые сохраняются в базе данных. Затем оператор, анализируя совокупность входных данных, введенных ранее пользователем, определяет соответствующее выходное состояние объекта. Таким образом, полученные знания накапливаются в базе данных, формируя базу знаний об объекте. Информация, хранящаяся в базе знаний, образует классы, определенные в соответствии с выходными состояниями объекта.

При втором способе (без участия пользователя) оператор, уже имея набор совокупностей входных данных об объекте, может загружать его в систему. Набор может быть уже проанализирован или впоследствии оценен экспертом.

Формат сообщений. Сообщения, передаваемые между компонентами системы и интерфейсной частью, состоят из конфиг-слов, разделенных символом «;». Каждое такое конфиг-слово представляет собой пару ключ–значение (*key=value*). Значение конфиг-слова может само являться конфиг-словом, в этом случае служебные символы («=» и «;») экранируются.

Например, параметры запуска компонента *DBInteractor* представляют собой набор конфиг-слов:

db=host\=127.0.0.1\;port\=5432\;login\=start\;password
=Rjj0Hvbel\;dbname\=start;
port=3428

Как можно видеть, здесь значение конфиг-слова с ключем *db* представляет собой набор конфиг-слов (*host, port, login, password, dbname*). Поскольку они содержатся в значении конфиг-слова, то служебные символы экранируются.

ПОЛЬЗОВАТЕЛЬСКИЕ ЭКРАНЫ

Авторизация и рабочее место. Авторизовавшись, пользователь попадает на свое рабочее место, где отображаются его личные данные, к которым относятся ФИО, логин, роль, город, e-mail. Также пользователь может видеть меню, в которое выводится список доступных функций (зависит от полномочий).

Функции комплекса могут быть разделены на несколько групп, функциональность которых определяется полномочиями пользователей.

Для осуществления авторизации *PHP*-скрипты передают на *UMServer*:

– *command*=auth;
– *user*=<номер подключения>;
– *login*=<логин>;
– *password*=<пароль>.

После выполнения команды *UMServer* возвращает:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=
  – *id*=<идентификатор пользователя>;
  – *name*=<логин пользователя>;
  – *password*=<хеш пароля пользователя>;
  – *firstName*=<имя пользователя>;
  – *middleName*=<отчество пользователя>;
  – *lastName*=<фамилия пользователя>;
  – *city*=<город пользователя>;
  – *email*=<электронная почта пользователя>;
  – *role*=
    – *id*=<права пользователя>;
    – *name*=<название группы пользователя>.

Управление пользователями. Группа управления пользователями содержит несколько функций:

1) изменение/возвращение личной информации (ФИО, пароль, город, электронная почта) – менять логин не разрешается;
2) добавление/удаления пользователей;
3) получение списка пользователей.

Для добавления нового пользователя администратор должен зарегистрировать его в системе. Для этого *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=setUser*;
– *user*=<номер подключения>;
– *name*=<логин пользователя>;
– *password*=<хеш пароля пользователя>;
– *firstName*=<имя пользователя>;
– *middleName*=<отчество пользователя>;
– *lastName*=<фамилия пользователя>;
– *city*=<город пользователя>;
– *email*=<электронная почта пользователя>;
– *role*=
  – *id*=<права пользователя>.

После выполнения команды *Web*-сервер получает следующий ответ:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=<присвоенный идентификатор>.

Изменение личных данных может быть осуществлено самим пользователем или администратором. Для изменения информации о пользователе *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=setUser*;
– *user*=<номер подключения>;
– *id*=<идентификатор пользователя>;
– *password*=<хеш пароля пользователя> – необязательный параметр;
– *firstName*=<имя пользователя>;
– *middleName*=<отчество пользователя>;
– *lastName*=<фамилия пользователя>;
– *city*=<город пользователя>;
– *email*=<электронная почта пользователя>;
– *role*= – необязательный параметр; роль может изменять только администратор
  – *id*=<права пользователя>.

После выполнения требуемых действий *UMServer* возвращает следующие параметры:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=<присвоенный идентификатор>.

Для возвращения информации о пользователе *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=getUser*;
– *user*=<номер подключения>;
– *id*=<идентификатор пользователя>.

UMServer возвращает:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=
  – *id*=<идентификатор пользователя>;
  – *name*=<логин пользователя>;
  – *password*=<хеш пароля пользователя>;
  – *firstName*=<имя пользователя>;
  – *middleName*=<отчество пользователя>;
  – *lastName*=<фамилия пользователя>;
  – *city*=<город пользователя>;
  – *email*=<электронная почта пользователя>;
  – *role*=
    – *id*=<права пользователя>;
    – *name*=<название группы пользователя>.

Для возвращения списка зарегистрированных в системе пользователей *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=getUsers*;

– *sort*=<сортировка> – возможные значения:
  – *id* – сортировка по идентификатору;
  – *name* – сортировка по логину;
  – *fname* – сортировка по имени;
  – *lname* – сортировка по фамилии;
  – *mname* – сортировка по отчеству;
  – *role* – сортировка по правам доступа;
  – *city* – сортировка по городу;
  – *email* – сортировка по электронной почте;
– *ascend*=<направление сортировки> – возможные значения: *asc* и *desc*;
– *user*=<номер подключения>.

Получив коллекцию пользователей *UMServer* возвращает следующие параметры:

1. *code*=<код ошибки>;
2. *text*=<текст ошибки>;
3. *value*=<массив пользователей> – каждый пользователь содержит следующие поля:

– *id*=<идентификатор пользователя>;
– *name*=<логин пользователя>;
– *password*=<хеш пароля пользователя>;
– *firstName*=<имя пользователя>;
– *middleName*=<отчество пользователя>;
– *lastName*=<фамилия пользователя>;
– *city*=<город пользователя>;
– *email*=<электронная почта пользователя>;
– *role*=
  – *id*=<права пользователя>;
  – *name*=<название группы пользователя>.

Для удаления пользователя из системы *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=removeUsers*;
– *id*=<идентификатор пользователя>
– *user*=<номер подключения>.

Произведя удаление, *UMServer* возвращает следующие параметры:

– *code*=<код ошибки>;
– *text*=<текст ошибки>.

Управление объектами моделирования. Группа управления объектами моделирования содержит следующие функции:

1) установка/возвращения объекта моделирования;
2) добавление/удаления объекта моделирования;
3) возвращение всех объектов системы;
4) возвращение всех объектов пользователя.

Для добавления объекта моделирования *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=setSimulationObject*;
– *name*=<название объекта>;
– *description*=<описание объекта>;
– *owner*=<идентификатор владельца>;
– *type*=
  – *id*=тип объекта;
– *dataNumber*=<минимальное количество строк обучающей выборки, необходимой для моделирования объекта>;
– *inputNumber*=<количество входных параметров>;
– *outputNumber*=<количество выходных параметров>;
– *user*=<номер подключения>.

Для возвращения числа входов и выходов объекта используется следующая команда:

– *command=getObjectInputNumber*;
– *id*=<идентификатор объекта>;
– *user*=<номер подключения>.

Ответ выглядит следующим образом:

– *code*=<код ошибки>;
– *text*=<текст ошибки>.

Для добавления новой строки обучающей выборки используется команда:

– *command=addPatternRow*;
– *value*=<массив значений> (формат массива следующий – числа должны быть разделены символом табуляции);
– *type*=<тип выборки> (1 – входная выборка, 0 – выходная);
– *user*=<номер подключения>.

В ответ приходит результат операции:

– *code*=<код ошибки>;
– *text*=<текст ошибки>.

Управление ИНС-моделями. Группа управления ИНС-моделями содержит следующие функции [3–4]:

– установка/возвращение ИНС-модели;
– добавление/удаление ИНС-модели;
– возвращение всех ИНС-моделей;
– возвращение всех ИНС-моделей конкретного пользователя;
– возвращение всех ИНС-моделей конкретного объекта;
– переобучение/дообучение ИНС-модели.

Для создания новой ИНС-модели на основе объекта пользователям необходимо передать на *UMServer* следующие параметры:

– *command=setANNModel*;
– *name*=<название модели>;
– *value*=<текст конфигурации>;
– *object*=<идентификатор моделируемого объекта>;
– *owner*=<идентификатор пользователя, создающего модель>;
– *user*=<номер подключения>.

UMServer возвращает следующие параметры:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=<идентификатор, присвоенный ИНС-модели>.

Для возвращения конкретной ИНС-модели *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=getANNModel*;
– *id*=<идентификатор модели>;
– *user*=<номер подключения>.

UMServer в ответ возвращает

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=
  – *id*=<идентификатор модели>;
  – *name*=<название модели>;
  – *value*=<текст конфигурации модели>;
  – *owner*=<владелец модели>;
  – *object*=<идентификатор моделируемого объекта>;
  – *trainedLevel*=
    – *id*=<уровень обученности>;
    – *name*=<название уровня обученности>;
– *user*=<номер подключения>.

Созданные ИНС-модели могут быть изменены. В этом случае старые конфигурации сохраняются в истории моделей. Для изменения ИНС-модели *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=setANNModel*;
– *id*=<идентификатор модели>;
– *name*=<название модели>;
– *value*=<текст конфигурации>;
– *object*=<идентификатор моделируемого объекта>;
– *owner*=<идентификатор пользователя, создающего модель>;
– *user*=<номер подключения>.

UMServer возвращает следующие параметры:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=<идентификатор ИНС-модели>.

Для возвращения ИНС-моделей пользователя *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=getANNModels*;
– *id*=<идентификатор пользователя>;
– *sort*=<сортировка> – возможные значения:
  – *id* =<сортировка по идентификатору>;
  – *name* =<сортировка по названию>;
  – *user* =<сортировка по пользователю>;
  – *trainedLevel* =< сортировка по уровню облученности>;
– *ascend*=<направление сортировки> – возможные значения: *asc* и *desc*;
– *user*=<номер подключения>.

В ответ *UMServer* возвращает

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=<массив моделей; каждая содержит:>
  – *id*=<идентификатор модели>;
  – *name*=<название модели>;
  – *value*=<текст конфигурации модели>;
  – *owner*=<владелец модели>;
  – *object*=<идентификатор моделируемого объекта>;
  – *trainedLevel*=
    – *id*=<уровень обученности>;
    – *name*=<название уровня обученности>.

Для удаления ИНС-модели *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=removeANNModel*;
– *id*=<идентификатор модели>;
– *user*=<номер подключения>.

UMServer возвращает

– *code*=<код ошибки>;
– *text*=<текст ошибки>.

Для возвращения ИНС-моделей конкретного пользователя *PHP*-скрипты передают на *UMServer* следующие параметры:

– *command=getUserANNModels*;
– *owner*=<идентификатор пользователя>;
– *user*=<номер подключения>.

UMServer возвращает ответ:

– *code*=<код ошибки>;
– *text*=<текст ошибки>;
– *value*=<массив ИНС-моделей> – каждая модель содержит следующие поля:
  – *id*=<идентификатор модели>;
  – *name*=<название модели>;
  – *value*=<текст конфигурации модели>;
  – *owner*=<владелец модели>;
  – *object*=<идентификатор моделируемого объекта>;
  – *trainedLevel*=
    – *id*=<уровень обученности>;
    – *name*=<название уровня обученности>;
– *user*=<номер подключения>.

РЕАЛИЗАЦИЯ СИСТЕМЫ СОГЛАСНО ТЕХНОЛОГИИ *SaaS*

Описание технологии *SaaS*. Как известно, приложения, реализованные согласно технологии *SaaS* (приложение как услуга), обеспечивают повсеместный и удобный сетевой доступ по требованию к общему пулу настраиваемых вычислительных ресурсов, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру.

Одним из главных преимуществ модели *SaaS* является значительное уменьшение расходов на инфраструктуру информационных технологий и возможность гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности приложения, реализованного согласно модели *SaaS* [5–6].

Описание протокола. Для передачи сообщений протокол использует кодировку *UTF-8*. При отправке сообщения передаваемая строка конвертируется в последовательность байт.

Далее для удобства записи можно представить последовательность байт в шестнадцатеричном виде следующим образом: 0001FFFF -> 0F. Здесь первая цифра – первая последовательность из четырех байт, вторая цифра – вторая последовательность из четырех байт и т. д.

Полученная последовательность разбивается на части по 254 байта, которые в дальнейшем будут называться «пакеты».

В случае, когда число байт в сообщении меньше 254 (или не кратно 254), их может не хватить для минимальной величины пакета в 254 байта. В этом случае пакет дополняется байтом, имеющим значение «*7C*». Таким образом, размер всех пакетов получаются равным 254 байта.

Для того чтобы принять сообщение, необходима информация о том, является ли пакет последним. Для этого каждому пакету в конец добавляется управляющий байт. Если пакет не является последним, то добавляется байт со значением «*2B*».

Если пакет является последним, то в конец добавляется байт со значением «*2D*».

Для того чтобы принять сообщение, необходимо склеить все пакеты в том порядке, в котором они приняты, и удалить управляющие байты (байты со значениями «*2B*» «*7C*» «*2D*»).

Далее эта последовательность байт интерпретируется как строка в кодировке *UTF-8*.

Реализация технологии SaaS. Для того чтобы передать сообщение серверу, необходимо выполнить авторизацию, для чего нужно отправить сообщение:

: user=client;message='';

В ответ клиент получит сообщение: «*user_id = = <идентификатор>*». После этого при каждой отправке клиент должен посылать сообщения следующего вида:

«user=client; user_id=<полученный при авторизации идентификатор>;message=<сообщение>»;

При авторизации комплекс отсылает сообщение вида:

user=complex;

После этого комплекс подключается к серверу и получает от него сообщения вида:

«*user=client; user_id=<полученный при авторизации идентификатор посылающего сообщение клиента>;message=<сообщение ползователя>*»;

После того как комплекс обработал сообщение клиента, он должен отослать обратно свой ответ, который имеет следующий вид:

«*user=client; user_id=<полученный при авторизации идентификатор отославшего сообщение клиента>;message=<сообщение комплекса>*» [7].

Реализация протолокла на стороне веб-сервера. На стороне веб-сервера реализация протокола осуществлена на языке *PHP*. Ниже приведен фрагмент исходного кода.

```
<?php
class Connection{
    var $socketIdentificator = -1;
    var $connectingFlag = false;
    var $serverFlag;
    var $address;
    var $port;
    var $socketBufferSize = 255;
    var $buf;
    var $error = 0;

    function Connection($address, $port){
       $this->address = $address;
       $this->port = $port;
       $this->connect();
    }

    function connect(){
        /*
        if ($this->serverFlag)//проверяем является
ли подключение клиентским
        {*/
//                              socket_close($this-
>socketIdentificator);
            $this->connectingFlag = false;
            $this->socketIdentificator = sock-
et_create(AF_INET, SOCK_STREAM, SOL_TCP);//создаем
новый сокет
            if($this->socketIdentificator < 0)
            {
                $this->error = "Error creat sock-
et";
                throw new Exception("Error create
socket", 1);
            }
            if($this->socketIdentificator       ===
false){
                $errorcode = socket_last_error();
                $errormsg = socket_strerror($error-
code);
                throw new Exception("Couldn't
create socket: [$errorcode] $errormsg", 1);
            }
            //создаем новое подключение
 //           print "11".$this->connectingFlag;
            if  (!socket_connect($this->socketIden-
tificator, $this->address, $this->port))
            {
                $this->error = "cannot connect";
                return false;
            }
            else
            {
                $this->connectingFlag = true;
                return true;
            }
        }
        else
        {
            String bufS = "подключение к клиенту со
стороны сервера невозможно";
            debug(bufS);
        }
    }


/**
* Строка в кодировке utf-8 преобразуется в массив
байт и подается на вход этой функции
* Отправка сообщения на вход подается массив байт
* функция strlen показывает не количество символов,
а количество байт
*/
    function sendMessage($message){
         $str = $message;
        $buf = "";
        for($i  = 0; $i<=strlen($str); $i+= $this-
>socketBufferSize -1){
            for($j    =    0;    $j    <    $this-
>socketBufferSize -1; $j++){
                if($i+$j >= strlen($str)){
                    $buf .= '|';
                } else{
                    $buf .= $str{$i+$j};//{} – воз-
вращение байта на определенной позиции строки
                }
```

```
            }
            if(($i+$this->socketBufferSize - 1) >=
strlen($str)){
                $buf{$this->socketBufferSize -1} =
'-';
            }else{
                $buf{$this->socketBufferSize -1} =
'+';
            }
            if($buf[0] != "|"){
                socket_send($this-
>socketIdentificator,          $buf,          $this-
>socketBufferSize, 0);
            }
            $buf = "";
        }
    }
/***
* Чтение сообщения
*       следующее обращени означает чтение байта
на определенной позиции - $str{2} в данном случае
3-го байта начиная с нуля
*/
    function recvMessage(){
        $stringBuf = "";
        $exit = false;
        while(!$exit){
            $buf = "";
            $bytesRead     =     socket_recv($this-
>socketIdentificator,          $buf,          $this-
>socketBufferSize, MSG_WAITALL);
            if(false) //($bytesRead <= 0)
            {
                //если не удалось прочитать часть
сообщения
                $bufS = "error read message";
                $exit = true;
                socket_close($this-
>socketIdentificator);
                $this->connectingFlag = false;
            }
            else{
                for  ($i  =  0;  $i<strlen($buf)-1;
$i++)
                {
                    if ($buf{$i} == '|')
                    {
                        break;
                    }
                    if(!(ord($buf{$i}) == 0)){
                        $stringBuf.=$buf{$i};
                    }
                }
                //если последний символ равен '-',
то сообщение последнее выходим из цикла while
                //если последний символ равен '+',
то сообщение не последнее продолжаем считывать
                if ($buf[strlen($buf)-1] == '+')
                {
                    $exit = false;
                }
                else
                {
                    $exit = true;
                }
            }
        }
        return $stringBuff;
    }

    function close(){
    }
}
```

Реализация протокола на стороне сервера приложений. На стороне сервера приложений реализация осуществлена на языке *C++*. Ниже приведен фрагмент сходного кода.

```
    /**
    *   Сокетное подключение
    */
    class Connection
    {
    public:
        /**
        *   Конструктор
        */
        Connection();
        /**
        *   Деструктор
        *   При наличии подключения оно завершается
        */
        ~Connection();
        /**
        *   Установка повторного подключения
        *   В случае уже имеющегося соединения про-
исходит переподключение
        */
        void connect();
        /**
        *   Отключение от сервера
        */
        void disconnect();
        /**
        *   Проверка наличия сооединения
        *   @return - флаг наличия подключения
        */
        bool isConnect() const;
        /**
        *   Отправка сообщения
        *   @param message - отправляемое сообщение
        */
        void  sendMessage(const  Message  message,
bool isComplex = true);
        /**
        *   Возвращение сообщения
        *    @return Message - возвращаемое сообще-
ние
        */
        Message  recvMessage(String  lastSimbol  =
EMPTY_STRING);
        /**
        *   Проверка наличия нового сообщения
        *   @return bool - флаг нового сообщения
        */
        bool hasMessage() const;
        /**
        *   Возвращение идентификатора
        *   @return int - идентификатор
        */
        int identificator() const;
    };

    /**
    *   Клиентское подключение к серверу
    */
    class ClientConnection : public Connection
    {
    public:
        /**
        *   Конструктор
        *   Устанавливает подключение к серверу
        *   @param idAdress - ip-адрес сервера
        *   @param port - порт, прослушиваемый сер-
вером
        */
        ClientConnection
            (
                String ipAdress = LOCAL_HOST_TEXT,
                const Port port = DEFAULT_PORT
            );
        /**
        *   Деструктор
        *   При наличии подключения оно завершается
        */
        ~ClientConnection();
        /**
        *   Установка повторного подключения к сер-
веру
        *   В случае уже имеющегося соединения про-
исходит переподключение
        *   По умолчанию параметры сервера текущие
        *   @param idAdress - ip-адрес сервера
        *   @param port - порт, прослушиваемый сер-
вером
        */
        void connect
```

```
            (
                const   String   ipAdress   =   LO-
CAL_HOST_TEXT,
                const Port port = DEFAULT_PORT
            );
    };

    /**
    *  Гнездо подключений
    */
    class ConnectionsNest
    {
    public:
        /**
        *   Конструктор
        *   @param port – прослушиваемый порт
        */
        ConnectionsNest(Port port = DEFAULT_PORT);
        /**
        * Конструктор
        * @param port – прослушиваемый порт
        * @param error – ошибка создания прослуши-
ваемого порта
        */
        ConnectionsNest(Port port, bool &error);
        /**
        *   Деструктор
        *   Завершение всех подключений
        */
        ~ConnectionsNest();
        /**
        *   Начало прослушивания
        *    @return bool – флаг определяющий по-
ставлен ли сокет на прослушку
        *   true – постановка на прослушку удалась
        *   false – постановка на прослушку не уда-
лась
        */
        bool listen();
        /**
        *   Возвращение нового подключения
        *   @return Connection * подключение
        */
        Connection * connection();
```

ЗАКЛЮЧЕНИЕ

Таким образом, разработана целостная интерактивная система, состоящая из взаимосвязанных компонентов, позволяющих осуществлять построение нейросетевых систем.

ЛИТЕРАТУРА

1. *Арзамасцев А.А., Зенкова Н.А., Неудахин А.В.* Автоматизированная технология построения экспертных систем с интеллектуальным ядром на основе ИНС-моделей // Открытое образование. 2008. № 3 (68). С. 35-39.
2. *Арзамасцев А.А., Зенкова Н.А., Крючин О.В., Квашенкин Д.О., Неудахин А.В.* Автоматизированная технология и программно-технологический комплекс для построения экспертных систем с интеллектуальным ядром, основанным на нейросетевых моделях, поддержкой распределенного ввода данных и параллельных вычислений // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2012. Т. 17. Вып. 3. С. 948-978.
3. *Крючин О.В., Арзамасцев А.А., Королев А.Н., Горбачев С.И., Семенов Н.О.* Универсальный симулятор, базирующийся на технологии искусственных нейронных сетей, способный работать на параллельных машинах // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2008. Т. 13. Вып. 5. С. 372-375.
4. *Крючин О.В.* Программный комплекс для моделирования объектов социально-экономического назначения с использованием искусственных нейронных сетей на кластерных вычислительных системах // Психолого-педагогический журнал Гаудеамус. Актуальные проблемы информатики и информационных технологий: материалы 14 междунар. науч.-практ. конф. Тамбов, 2010. № 2 (16). С. 534.
5. *Martin R.J., Hoover N.* Guide To Cloud Computing // Information Week. 2008.
6. *Терехов И.* Не рано ли Cloud Computing в массы? // Компьютерра. 2009. 19 окт.
7. *Крючин О.В.* Информационная система и функционирование ее компонентов согласно технологии SaaS // Электронный научный журнал «Исследовано в России» – 009/130527. С. 131-146. URL: http://zhurnal.ape.relarn.ru/articles/2013/009.pdf. (дата обращения: 19.03.2014.)

Поступила в редакцию 25 марта 2014 г.

Kryuchin O.V., Arzamastsev A.A., Vyazovova E.V. INTERACTIVE INFORMATION SYSTEM WITH INTELLIGENT CORE BASED ON ARTIFICIAL NEURAL NETWORKS WITH WEB-INTERFACE

The functioning principles of interactive information system with the core based on artificial neural network technology are described. The system interface description and the protocol passing components messages are given. The sources code fragments of the core and interface synchronization module are presented.

*Key words***:** information system; artificial neural networks; protocol data passing.

Крючин Олег Владимирович, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, магистрант по направлению подготовки «Прикладная математика и информатика» института математики, физики и информатики, e-mail: kryuchov@gmail.com

Kryuchin Oleg Vladimirovich, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Candidate for Master's Degree of Direction of Preparation of "Applied Mathematics and Informatics" of Mathematics, Physics and Informatics Institute, e-mail: kryuchov@gmail.com

Арзамасцев Александр Анатольевич, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, доктор технических наук, профессор, зав. кафедрой компьютерного и математического моделирования, e-mail: arz_sci@mail.ru

Arzamastsev Alexander Anatolyevich, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Doctor of Technics, Professor, Head of Computer and Mathematical Simulation Department, e-mail: arz_sci@mail.ru

Вязовова Елена Владимировна, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, магистрант по направлению подготовки «Прикладная математика и информатика» института математики, физики и информатики, e-mail: kafedra_kmm@mail.ru

Vyazovova Elena Vladimirovna, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Candidate for Master's Degree of Specialty of "Applied Mathematics and Informatics" of Mathematics, Physics and Informatics Institute, e-mail: kafedra_kmm@mail.ru