УДК 004.822+622.276.054

ЗАСТОСУВАННЯ МОВИ ПРОГРАМУВАННЯ PYTHON ДЛЯ ПОБУДОВИ БАЗ ЗНАНЬ З ПРОБЛЕМ НАДІЙНОСТІ І ДОВГОВІЧНОСТІ ШТАНГОВИХ СВЕРДЛОВИННИХ НАСОСНИХ УСТАНОВОК

В.Б. Копей, І.І. Палійчук

ІФНТУНГ, 76019, м .Івано-Франківськ, вул. Карпатська, 15, тел. (03422) 43024,
e-mail: v k o p e y @ g m a i l . c o m

На основі фреймової моделі подання знань та універсальної мови програмування Python запропоновано принципи розробки баз знань та експертних систем з проблем надійності і довговічності штангових свердловинних насосних установок. Розглянуто можливості Python для подання знань. Розроблено демонстраційну базу знань та подано приклади запитів до неї.

Ключові слова: база знань, подання знань, експертна система, мова програмування Python, штангова свердловинна насосна установка.

На основе фреймовой модели представления знаний и универсального языка программирования Python предложены принципы разработки баз знаний и экспертных систем по проблемам надежности и долговечности штанговых скважинных насосных установок. Рассмотрены возможности Python для представления знаний. Разработана демонстрационная база знаний и даны примеры запросов к ней.

Ключевые слова: база знаний, представление знаний, экспертная система, язык программирования Python, штанговая скважинная насосная установка.

On the basis of frames for knowledge representation and general-purpose programming language Python the principles of development of knowledge bases and expert systems about the reliability and durability of sucker rod pumping unit have been proposed. The capabilities of Python for knowledge representation have been examined. The demo of knowledge base and the examples of querying to it have been developed.

Keywords: knowledge base, knowledge representation, expert system, programming language Python, sucker rod pumping unit.

1. Вступ

Відомо, що розвиток науки базується на отриманні об'єктивних і системно-організованих **знань**, які допомагають людям раціонально організувати свою діяльність і вирішувати різні проблеми, котрі виникають в її процесі. Сучасні комп'ютерні технології зумовили розвиток **інженерії знань** – області науки про **штучний інтелект**, яка вивчає методи і засоби отримання, подання, структурування і використання знань. Інженерія знань пов'язана з розробкою баз знань і експертних систем [1,2].

База знань - це сукупність **фактів** і **правил логічного висновку (виведення)** в обраній предметній області. **Правила логічного висновку** - це правила перетворення вихідної системи фактів (суджень) в нову систему фактів (висновків). Наприклад, за такими правилами з вихідних фактів "всі люди смертні" і "Сократ - людина" можна зробити висновок "Сократ смертний". В даному випадку правилом висновку є параметризоване твердження "якщо всі люди смертні і X – людина, то X смертний". Власне наявність правил висновку є основною відмінністю баз знань від баз даних.

Програма, яка виконує логічний висновок з попередньо побудованої бази фактів і правил у відповідності з законами формальної логіки, називається **машиною виведення.** База знань і машина виведення є основними компонентами **експертної системи** - програми, яка призначена для пошуку способів вирішення проблем з певної предметної області. Для такого пошуку користувач подає відповідні **запити до бази знань**.

Для збереження знань в комп'ютерній системі з можливістю їх подальшої автоматизованої обробки використовують різноманітні методи **подання (представлення) знань**. Прикладами методів подання **чітких знань** є: логіка першого порядку, мова програмування Пролог, реляційні системи, продукційна модель, фреймова модель, семантичні мережі [1,2]. Ці методи можна реалізувати різними формальними мовами подання знань, наприклад класу онтологій. Під **онтологією** в інформатиці розуміють детальну формалізацію деякої області знань за допомогою концептуальної схеми. Онтологія може використовуватись для представлення в базі знань ієрархії понять і їх відносин. Наприклад, мова Веб-Онтологій OWL (ontology web language) [3] була розроблена для реалізації концепції семантичної павутини в межах всесвітньої павутини (WWW) і є розширенням мови RDF (Resource Description Framework) - мови розмітки на основі XML, яка використовується для подання тверджень про ресурси в вигляді, придатному для машинної обробки. Діалекти OWL Lite і OWL DL основані на **дескріпційних логіках** – мовах подання знань, що дозволяють описувати поняття предметної області в недвозначному, формалізованому вигляді. Для створення запитів до даних, поданих за моделлю

RDF, використовується мова запитів SPARQL. Не зважаючи на усі переваги, OWL і SPARQL є спеціальними мовами, що певною мірою обмежує їх можливості.

2. Постановка задачі

На даний час накопичено багато знань, пов'язаних з проблемами надійності і довговічності штангових свердловинних насосних установок (ШСНУ). Проте існує проблема ефективного використання цих знань людьми, які займаються проектуванням чи обслуговуванням ШСНУ. Література і інші джерела експертної інформації з проблем надійності і довговічності ШСНУ в основному містять множину наукових фактів у вигляді причинно-наслідкових зв'язків та залежностей між факторами, які впливають на надійність ШСНУ. Виникає завдання ефективного подання таких знань. Основною ідеєю авторів є застосування в якості мови **подання знань** та **створення запитів** до бази знань мови програмування Python [4,5], яка, на відміну від спеціальних мов, є мовою загального призначення, і тому дає розробнику експертної системи набагато більше можливостей. Для цього можна використати **об'єктно-орієнтовані можливості** Python та її **засоби інтроспекції**, адже відомо, що такі методи подання чітких знань, як фрейми і семантичні мережі, можна легко реалізувати засобами **об'єктно-орієнтованого програмування** (ООП).

3. Огляд можливостей Python для подання знань

Чому авторами була вибрана саме Python? Python - розповсюджена високорівнева мова загального призначення з акцентом на продуктивність розробника. Основні її особливості: працює майже на усіх відомих платформах, є відкритим і вільним програмним забезпеченням, виконується шляхом інтерпретації байт-коду, підтримує кілька парадигм програмування (в тому числі об'єктно-орієнтоване), код програм компактний і легко читається, мові характерні динамічна типізація, повна інтроспекція, зручні структури даних (кортежі, списки, словники, множини), велика стандартна бібліотека та велика кількість сторонніх бібліотек різноманітного призначення. Інтерпретатор Python має інтерактивний режим роботи, при якому введені з клавіатури оператори відразу ж виконуються, а результат виводиться на екран. Незважаючи на відносно малу швидкодію програм Python знайшов своє застосування в різноманітних мобільних платформах (зокрема в Windows CE, Symbian OS і Android).

Python володіє потужними об'єктно-орієнтованими можливостями, наприклад, усі значення в програмі є об'єктами [5]. ООП основане на використанні **об'єктів**, які є абстрактними моделями реальних об'єктів. Об'єкти створюються за допомогою спеціальних типів даних - класів. Кожен **клас** описує множину об'єктів певного типу.

Python підтримує повну інтроспекцію часу виконання. Тобто для будь-якого об'єкта під час виконання можна отримати всю інформацію про його структуру [4]. Наприклад, найбільш відомим інструментом для інтроспекції в Python є функція `dir()`, яка повертає список імен атрибутів переданого їй об'єкта. Функція `type()` або атрибут `__class__` дозволяють отримати тип об'єкта. Функція `vars()` або атрибут `__dict__` дозволяють отримати словник з парами атрибут:значення об'єкта. Функції `hasattr()`, `getattr()` і `setattr()` дозволяють відповідно перевірити наявність у об'єкта заданого атрибута, повернути його і змінити значення. Функція `issubclass()` дає змогу визначити чи успадковується один клас від іншого, а метод `__subclasses__()` повертає список підкласів. Кортеж базових класів та їх ієрархію можна отримати за допомогою атрибутів `__bases__` і `__mro__`. Модуль `inspect` містить додаткові функції, які допомагають отримати інформацію про об'єкти під час виконання. Застосування інтроспекції дозволяє ефективно програмувати механізми виведення та запити до бази знань.

4. Приклад бази знань і запитів до неї мовою Python

Подамо приклад створення мовою Python демонстраційної бази знань, яка містить поняття відмов ШСНУ та факторів, які впливають на ці відмови, і причинно-наслідкові зв'язки між цими поняттями. Для побудови бази знань застосуємо фреймову модель подання знань.

Елементи розробленої авторами онтології частково відповідають основним елементам OWL Lite [3, 6]: класи (з можливістю створення підкласів), властивості (які можуть бути функціональними, інверсними, симетричними і транзитивними) і індивіди. Тому можливим є імпорт і експорт таких онтологій в OWL. Модуль Python, який містить базу знань і запити до неї повинен мати блочну структуру - спочатку створюються класи, потім індивіди, потім задаються властивості індивідів, а в кінці виконуються запити.

Класи онтології відповідають класам мови Python, а підкласи можна створювати за допомогою механізму успадкування Python. Так розроблено базовий клас `Base` (описує усі об'єкти, які мають ім'я) та класи, які успадковують його: `Factor` (описує фактор, який впливає на надійність СШНУ), `Fact` (описує факт у вигляді триплету суб'єкт-предикат-об'єкт), `Reference` (описує посилання на джерело факту), `Dependence` (описує залежність величини X від величини Y).

```
# -*- coding: UTF-8 -*-
class Base(object):
    '''Базовий клас онтології'''
    def __init__(self,name): # конструктор
        self.name=name # назва об'єкта
        # ці властивості дозволяють створювати нові поняття
        # за допомогою логічних операцій
        Property(subj=self,name='And') # властивість 'And'
        Property(subj=self,name='Or') # властивість 'Or'
        Property(subj=self,name='Not') # властивість 'Not'
        if self.name not in KB.keys(): # якщо назви немає в базі
            KB[self.name]=self # добавити себе в базу знань

class Factor(Base): # успадковує клас Base
    '''Клас, який описує фактор'''
    def __init__(self,name): # конструктор
        Base.__init__(self,name) # виклик конструктора базового класу
        # транзитивна властивість 'є причиною'
        Property(subj=self, name='isCause',
                 inverseName='isEffect', transitive=True)
        # транзитивна властивість 'є наслідком'
        Property(subj=self, name='isEffect',
                 inverseName='isCause', transitive=True)

class Fact(Base):
    '''Клас, який описує факт (триплет) у вигляді
    суб'єкт-предикат-об'єкт'''
    def __init__(self,subjName,propName,objName): # конструктор
        self.name=subjName+'.'+propName+'.'+objName # назва об'єкта
        Base.__init__(self,self.name) # виклик конструктора базового класу
        self.subjName=subjName # назва суб'єкта
        self.propName=propName # назва предиката (властивість)
        self.objName=objName # назва об'єкта
        # добавити значення в властивість, якщо немає
        KB[subjName].__dict__[propName].add(KB[objName])
        # властивість 'має посилання'
        Property(subj=self,name='hasReference',inverseName='isReference')
        # властивість 'має залежність'
        Property(subj=self,name='hasDependence',inverseName='isDependence')

class Reference(Base):
    '''Клас, який описує посилання на джерело'''
    def __init__(self,name): # конструктор
        Base.__init__(self,name)
        # властивість 'є посиланням'
        Property(subj=self,name='isReference',inverseName='hasReference')

class Dependence(Base):
    '''Клас, який описує залежність'''
    def __init__(self, name, xy, relative=None, xName='x', yName='y'):
        ...
    def plot(self):
        '''Рисує графік залежності'''
        ...
    def interp(self,x,reverse=False):
        '''Знаходить значення лінійною інтерполяцією'''
        ...
```

Індивіди онтології відповідають об’єктам Python – значенням словника `KB`. Такий спосіб дозволяє звертатись до індивідів за їх іменем у вигляді Юнікод-рядка. Індивіди можуть мати властивості у вигляді атрибутів Python, які дозволяють описувати відношення між індивідами. Властивості є об’єктами класу `Property`, мають атрибути `subj` (суб'єкт властивості - це індивід, який має дану властивість), `name` (назва властивості), `inverseName` (назва інверсної властивості), `functional` (визначає чи властивість функціональна), `symmetric` (визначає чи властивість симетрична), `transitive` (визначає чи властивість транзитивна), `set` (множина значень властивості) та методи `__init__()` (конструктор, створює властивість), `add()` (добавляє об'єкт або кортеж об'єктів в множину) і `__call__()` (повертає множину значень властивості). Останні два методи реалізують елементи машини виведення. Такі елементи можуть бути присутні також в запитах до бази знань.

```
class Property(object):
    '''Клас, який описує властивість об'єкта'''
    def __init__(self, subj, name, inverseName='',
                 functional=False, symmetric=False, transitive=False):
        '''Конструктор'''
        self.subj=subj # суб'єкт властивості
        self.name=name # назва властивості
        self.inverseName=inverseName # назва інверсної властивості
        self.functional=functional # властивість функціональна
        self.symmetric=symmetric # властивість симетрична
        self.transitive=transitive # властивість транзитивна
        self.set=set() # множина значень властивості
        # установити атрибут властивості для суб'єкта
        self.subj.__setattr__(self.name,self)
    def add(self,*args):
        '''Добавляє об'єкт або кортеж об'єктів в множину'''
        ...
    # повертає множину значень властивості
    def __call__(self, showTransitive=False):
        '''Повертає множину значень властивості
            (showTransitive=True - транзитивної властивості)'''
        ...
```

База знань містить індивіди "втома", "корозія", "концентрація напружень", "циклічне навантаження", "агресивне середовище" і т.д. - об'єкти відповідних класів. Створений об'єкт індивіда автоматично додається в словник KB. Блок створення об'єктів індивідів виглядає так:

```
KB={} # словник бази знань
Factor('втома') # створити об'єкт-фактор 'втома'
Factor('корозія')
Factor('концентрація напружень')
Factor('циклічне навантаження')
Factor('агресивне середовище')
Factor('зношування')
Factor('тертя')
Factor('тертя і корозія')
Reference('Оптимізація параметрів різьбових з'єднань: Звіт про НДР /
В.Б.Копей, В.В.Михайлюк, Н.В.Шатинський / ІФНТУНГ.-2010.-143с.')
Dependence(name='Залежність екв. напруж. від велич. рад. скругл. зарізьбової
канавки', xy=[(3.0,675),(3.2,620),(3.4,630),(3.6,650),(3.8,690)], relative='є
мінімум')

# блок створення фактів
# суб'єкт 'концентрація напружень' 'є причиною' об'єкта 'втома'
Fact('концентрація напружень','isCause','втома')
```

Клас `Base` містить атрибут `name` (ім'я об'єкта) та властивості `And`, `Or`, `Not`, які дозволяють створювати нові об'єкти за допомогою логічних операцій. Клас `Factor` містить також властивості `isCause` (є причиною) і `isEffect` (є наслідком). Клас `Fact` містить атрибути `subjName` (назва суб'єкта), `propName` (назва предиката - властивості), `objName` (назва об'єкта) та властивості `hasReference` (має посилання), `hasDependence` (має залежність). Клас `Reference` містить властивість `isReference` (є посиланням). Клас `Dependence` містить атрибути `xy` (дані залежності), `relative` (відносний характер залежності), властивість `isDependence` (є залежністю) та функції `plot` (рисує графік залежності) і `interp` (знаходить значення лінійною інтерполяцією).

Властивість створюється під час створення індивіда шляхом виклику її конструктора `__init__()` з параметрами `subj`, `name`, `inverseName`, `functional`, `symmetric`, `transitive`, які описують її характеристики. Так властивості `isCause` і `isEffect` є взаємно **інверсними**. Це дозволяє, наприклад, з твердження "циклічне навантаження є причиною втоми" робити логічний висновок "втома є наслідком циклічного навантаження". Вони також є **транзитивними**. Це дозволяє, наприклад, з тверджень "шорсткість є причиною концентрації напружень" і "концентрація напружень є причиною втоми" робити логічний висновок "шорсткість є причиною втоми". Властивості можуть бути **функціональними**, тобто мати унікальне значення, и **симетричними**, що дозволятиме, наприклад, з твердження "X є Y" робити логічний висновок "Y є X".

Значення в властивість добавляються за допомогою методу `add()`. Блок додавання значень до властивостей індивідів виглядає так:

```
# 'тертя' 'є причиною' 'зношування'
KB['тертя'].isCause.add(KB['зношування'])
KB['циклічне навантаження'].isCause.add(KB['втома'])
KB['корозія'].isEffect.add(KB['агресивне середовище'])
KB['шорсткість поверхні'].isCause.add(KB['зношування'],KB['концентрація
напружень'])
KB['тертя і корозія'].And.add(KB['тертя'],KB['корозія'])
KB['концентрація напружень.isCause.втома'].hasReference.add(KB['Оптимізація
параметрів різьбових з'єднань: Звіт про НДР / В.Б.Копей, В.В.Михайлюк,
Н.В.Шатинський / ІФНТУНГ.-2010.-143с.'])
KB['концентрація напружень.isCause.втома'].hasDependence.add(KB['Залежність
екв. напруж. від велич. рад. скругл. зарізьбової канавки'])
```

Запити до бази знань створюються шляхом пошуку об'єктів в множинах. Для цього можна використовувати оператори Python `for` та `if`. Для доступу до множин використовують об'єкт `KB`, метод класу `Property __call__()` та стандартні математичні операції над множинами. Метод `__call__()`з параметром `showTransitive=True` повертає множину усіх значень транзитивної властивості. Наприклад, запит, який виводить усі причини фактора "втома":

```
for x in KB['втома'].isEffect(True):
        print x.name+' |',
```

Наступний запит виводить факти, які пов'язані з заданим джерелом:

```
for x in KB['Оптимізація параметрів різьбових з'єднань: Звіт про НДР /
В.Б.Копей, В.В.Михайлюк, Н.В.Шатинський / ІФНТУНГ.-2010.-
143с.'].isReference():
        print x.name+'('+x.subjName,x.propName,x.objName+')|'
```

Або складніший запит, який виводить джерела, які пов'язані з фактором "втома" та містять в назві рядок "різьб":

```
# обмежимо множини пошуку
p=[]
m=[]
for x in KB.itervalues():
    # якщо клас об'єкта 'Reference' і в назві є рядок 'різьб'
    if x.__class__.__name__=='Reference' and x.name.find('різьб')!=-1:
        p.append(x) # добавити в список p
    if x.__class__.__name__=='Fact': # якщо клас об'єкта 'Fact'
        if x.objName=='втома' or x.subjName=='втома':
            m.append(x) # добавити в список m
for x in p: # для усіх об'єктів списку p
    for y in m: # для усіх об'єктів списку m
        if x in y.hasReference(): # якщо x є серед y.hasReference()
            print x.name # вивести назву x
```

На відміну від OWL, в класах такої онтології можуть бути властивості, які не зберігають конкретного значення, а дозволяють його обчислити. Наприклад, метод класу `Dependence interp()` за заданим значенням X обчислює значення Y шляхом лінійної інтерполяції.

```
# список залежностей факту 'концентрація напружень.isCause.втома'
dep=list(KB['концентрація напружень.isCause.втома'].hasDependence())
print dep[0].interp(3.3) # вивести значення Y шляхом інтерполяції
print dep[0].interp(660,reverse=True) # вивести значення X шляхом
інтерполяції
dep[0].plot() # побудувати першу залежність
```

Перед блоком запитів можна створити блок з правилами виведення. Для прикладу, створимо правило виведення: "якщо Y є значенням властивості `Or` X і X є причиною Z, то Y теж є причиною Z".

```
for x in KB.itervalues():
    if x.__class__.__name__=='Factor': # для усіх факторів
        for y in x.Or(): # для усіх об'єктів в властивості Or об'єкта x
            for z in x.isCause(): # для усіх об'єктів в властивості isCause
об'єкта x
                # добавити нові факти в базу знань
                y.isCause.add(z) # y є причиною z
```

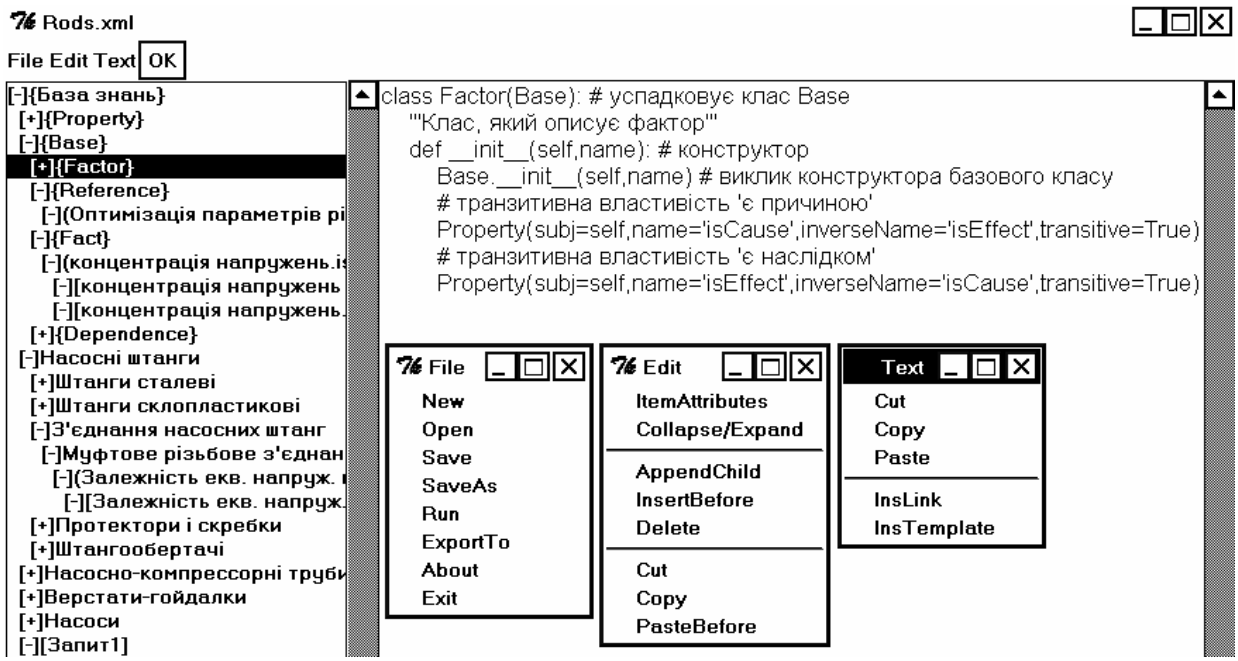

Рисунок 1 – Графічний інтерфейс експертної системи

Наявність великої кількості таких правил може суттєво уповільнити виконання програми, тому бажано виконувати цей код тільки один раз і зберігати об'єкти бази знань в постійній пам'яті за допомогою способів серіалізації даних Python (модулів `pickle` або `shelve`).

Текстовий режим роботи з таким модулем створює певні незручності створення великої бази знань. Тому для ефективного використання програма потребує надбудови у вигляді графічного інтерфейсу користувача (GUI), наприклад, розробленої авторами програми з GUI на основі Tkinter [5]. Tkinter – стандартна бібліотека Python на основі Tcl/Tk для розробки кросплатформних програм з GUI, яка містить мінімальний набір компонентів для побудови GUI, проте дуже проста у застосуванні. Інтерфейсом (рис. 1) розроблена програма подібна на традиційний аутлайнер (outliner) – програму, яка дозволяє організувати текст в деревовидну структуру або ієрархію. Справа у вікні програми знаходиться дерево понять з класами, індивідами, скриптами (тобто іншим кодом Python) та Python-коментаріями. Тут класи позначаються дужками {}, індивіди – (), скрипти – []. Зліва подається Python-код вибраного поняття. За допомогою команди Python `exec`, яка забезпечує динамічне виконання коду Python, можна виконати весь код бази знань разом з запитами до неї (меню File/Run). Програма є кросплатформною (працює у Windows, Linux та Windows CE), написана на чистому Python 2.6 з використанням стандартних бібліотек та має низку додаткових можливостей, таких як експорт бази знань у зовнішній файл, гіперпосилання та вставка шаблонів коду.

5. Висновки

Запропоновані авторами методи побудови баз знань і експертних систем мовою Python мають переваги у порівнянні з існуючими. Основною перевагою є те, що розробнику доступні усі широкі можливості мови Python. Під час розширення функціональності системи немає потреби винаходити ще одну спеціальну мову опису знань і мову запитів до них. Завдяки універсальності Python база знань є гнучкою до змін, існує можливість легкого удосконалення класів, атрибутів, правил виведення і запитів. Ці принципи можна використати для розробки повноцінних експертних систем в науці і техніці.

В такому підході до побудови баз знань є і певні недоліки. Зокрема, користувач повинен добре володіти мовою Python. Недоліком є також складність програмування машини виведення. Проте, якщо онтологія може бути експортована в OWL, то до програми можна підключати відомі машини введення FaCT++, HermiT, Cwm (реалізована на Python), Pellet, які мають ефективні алгоритми і доступні у вихідному коді.

Література

1 **Субботін С.О.** Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навчальний посібник / С.О. Субботін. – Запоріжжя: ЗНТУ, 2008. – 341 с.

2 **Рыбина Г.В.** Основы построения интеллектуальных систем / Г.В. Рыбина. – М.: Финансы и статистика, ИНФРА-М, 2010. – 432 с.

3 OWL Web Ontology Language. Overview [Електронний ресурс]: W3C Recommendation 10 February 2004 / W3C. – Режим доступу: http://www.w3.org/TR/owl-features/

4 **Бизли Д.** Python. Подробный справочник / Дэвид Бизли. – СПб.: Символ-Плюс, 2010. – 864 с.

5 **Лутц М.** Программирование на Python / Марк Лутц. – СПб.: Символ-Плюс, 2002. – 1136 с.

6 **Копей В.Б.** Принципи розробки бази знань з проблем надійності і довговічності різьбових з'єднань / В.Б. Копей, Ю.Д. Петрина // Науковий вісник Національного технічного університету нафти і газу. – № 4(26). – 2010. – С.66-69.

Стаття надійшла до редакційної колегії 09.11.11
Рекомендована до друку професором ***Ю. Д. Петриною***