Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 21 Issue 1 Version 1.0 Year 2021
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Green Computing using Perl and Python

By Poli Venkata Subba Reddy & Srivibha Vadravu
Sri Venkatesswara University

Abstract- Green computing or clean computing is necessary for Software Engineering. Perl and Python are important programming languages for green computing. Perl is regular language. Perl is mainly used for server-side programming because it is a regular language. It a portable and green programming language. It can be used as object-oriented (OO) or non object-oriented (Non-O-O) programming language. Python is preprocessor. It is a portable language for software engineering. It has an import feature for Green computing.

Keywords: *green computing, regular expressions objectoriented, perl, preprocessor, python.*

GJCST-C Classification: *K.7*

Strictly as per the compliance and regulations of:

Green Computing using Perl and Python

Poli Venkata Subba Reddy [α] & Srivibha Vadravu [σ]

Abstract- Green computing or clean computing is necessary for Software Engineering. Perl and Python are important programming languages for green computing. Perl is regular language. Perl is mainly used for server-side programming because it is a regular language. It a portable and green programming language. It can be used as object-oriented (OO) or non object-oriented (Non-O-O) programming language. Python is preprocessor. It is a portable language for software engineering. It has an import feature for Green computing.

Keywords: green computing, regular expressions object-oriented, perl, preprocessor, python.

I. Introduction

Green Computing or Clean Computing is necessary to solve different types of problem solving. The Programming is needed portability of the code, and less computation time. There are different techniques methods are used for Green Computing like recursion, parallelism, regular expression and Object-Oriented. The recursion is the calling function itself. Parallelism is computing the number of tasks at a time. The regular expression is simplifies the code. The Object-Oriented shall made program is independent.

II. Green Computing Methods

Programming is the main component for problem solution. The Green programming has some of the main features.

Portability
Less computing time
Reusability
Green Computing maybe studied with three methods.

a) Analysis

There are different analysis methods. Mainly

Time Complexity
Space Complexity

b) Design

There are different design method are used f

Divide and Conquer
Object-Oriented
Component based

c) Coding

The Programming Languages fall under different paradigms Imperative, Functional, Logical, and Object-Oriented and regular it is difficult to learn all the programming languages. It easy to learn programming languages through common principles like iteration, recursion, control statements, functions, functions, subroutines, Object-oriented, etc. All principles and techniques are not available in single programming language. The selected Programming Languages are discussed for Green Computing.

Programming languages are designed based on Automata. Context-Free Language is the recursively representation of Finite Automata.

For green computing, Recursive Algorithms and Parallel algorithms are used until recently. Programming languages are playing a main role.

We consider Perl and Python for green computing. Perl is the regular language. It simplifies the programming, and it reduces time.

Python is the preprocessing language. It simplifies the with the import feature, it simplify the code.

III. Component Technology

All components are specialized, independently deployed and extendable for the product. These components are also extendable to multi versions of the components. The following are the characteristics of the components.

The components have an externally accessible view.

The semantics such as business rules and regulations are defined for the composition of components.

As Component software extended, the components are extendable.

The component must be relocate and replace a component for other implantations or the development of new software system.

The semantic primitives must be extendable to new components.

The composition of components is tightly coupled.

The components are substituted and integrated into the other systems. Sometimes this maybe referred to as off-the-components.

a) Component Architecture

The component architecture mainly consists of Conceptual component model, infrastructure technologies and structured domain concepts. The

Author α: Department of Computer Science and Engineering, College of Engineering, Sri Venkatesswara University, Tirupati-517502, India.
e-mail: vsrpoli@hotmail.com
Author σ: Software Engineer, InfoSys, Hyderabad.
e-mail: vadrevu@gmail.com

component architecture comes across distributed, heterogeneous and new infrastructure technology. Integrated component architecture is the mechanism universal Component architecture and it may be referred to integration of independent component architectures. The integration may be loosely coupled and tightly coupled. It Describes implementation of Component infrastructure, Structured conceptual model, and domain concepts.

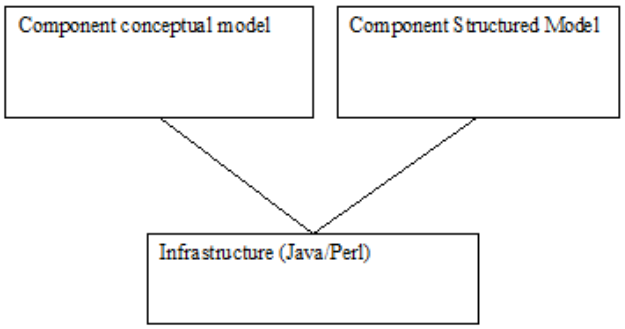

Figure 1: Component Architecture

b) Component Implementation

The component model is translated into component ware with tools for automation and management of components and interfaces. Interface to understand system architecture with the interface specifications that implement, reuse, and replacement of components. They are two types of component ware implementation for products.

Self-development in which component were developed from the scratch.

Off-the-self components in which component ware developed by black box assembling commercially available components and such components are documented, assembled and adapted.

The following are the characteristics of the implementation enterprise model. The components of the product may represent entire system

Generosity: It is stepwise instantiation and controlled processes that use specifications, inheritance, relationships and contexts.

Domain system: It represents a particular area of components.

Domain object: It represents a particular process of components.

Semantic primitives: These are rules and kinds of relationships between objects.

These domain concepts are used to compose domain components of individual components.

IV. Perl Programming

The Programming Languages fall under different paradigms Imperative, Functional, Logical, and Object-Oriented and Regular. It is difficult to learn all the programming languages. It made easy to learn programming languages through common principles like iteration, recursion, control statements, functions, functions, subroutines, Object-oriented etc. All principles and techniques are not available in single programming language. The selected Programming Languages are discussed for Green Computing.

The Programming Languages are constructed mainly based on Finite Automata (FA) and Regular (RE).

The Formal Languages (FL) are simple representation of Context-Free Language) CFL). The CFL is recursion of FA.

$$FA\ M = \{\Sigma, Q, \delta, q0, F\}$$

For instance,

$$\Sigma = a\text{-}z(a\text{-}z, 0\text{-}9)$$

$$Id = \{num, num1, x1, \ldots\}$$

Regular

$$M = \{\Sigma^*, Q, \delta, q0, F\}$$

$$\Sigma^* = \{a\text{-}z, 0\text{-}9\}^*$$

$$Id = a\text{-}z(A\text{-}Z, 0\text{-}9)^*$$

$$Id = \{x, x11, num, sum, sum12, \ldots\}$$

The CFL is defined as

$$M = < V, T, P, S >$$

$$E = E+E/E-E/E*E/id$$

The grammar

$$G = \{ A \Diamond \alpha w\}, \text{ where } \alpha \epsilon V, w \epsilon \{ N \cup \Sigma\}$$

The regular grammar

$$G^* = \{ A \Diamond \alpha w^*\}, \text{ where } \alpha \epsilon V, w^* \epsilon \{ N \cup \Sigma^*\}$$

For instance,

$$\Sigma = a\text{-}z(a\text{-}z, 0\text{-}9)$$

$$Id = num, id = x1 \text{ etc.}$$

$$\Sigma^* = \{a\text{-}z, 0\text{-}9\}^*$$

$$Id = a\text{-}z(A\text{-}Z, 0\text{-}9)^*$$

$$Id = \{x, x11, num, sum, sum12, \ldots\}$$

Perl is only Regular Language. Perl can be used as Non-Object-Oriented and Object-Oriented. Perl is Portable because like Algorithmic language. Perl is green programming longuge. The main concepts of Perl are recursion, regular, parallelism and client/server.

a) Recursion

Recursion is calling function by itself.

For instance,

```
$n=<STDIN>;
$factorial=fact($n);
print "$factorial\n";
```

```
sub  fact($num)
{
 if ($num==1) {return 1;}
else { return $num*fact($num-1);}
}
```

b) Regular

A regular expression is simply Expression of Finite Automate.

Consider the Regular Expression
Id =digit*. digit)+

Regular expressions are used to match the pattern, sting with

"m//", "s///", "qr//" and "split" operators
Simple string matching

c) Object-Oriented

Object are often called instance data or object attributes, and data fields

```
sub teacher ::pvsr{
  print "teaching dbms\n";
}
sub student::syam{
  print "tacking dbms course \n";
}
sub room::cse201{
  print "course in a201\n"
}
teacher::pvsr;
student::dbms;
room:cse201;
```

"Class->method" invokes subroutine "method" in package "Class "

```
teacher->dbms;
student>dbms;
room->cse201;
```

d) Threads

The "use thread" creates one or more threads.

```
 use threads;
$thr1= threads->new(\&ascending);
$thr2= threads->new(\&decending);
my $num ;
sub ascending {
    my $num;
 while ( 10)
   print " $num++\n";
}
sub decending {
   my $num=10;
 while (  0)
   print " $num--\n";
}
$thr1-> join;
$thr2->join;
```

e) Client/Server

Perl is powerful server side programming language because Perl is the only regular language.
For instance,
Computing two numbers at sever side
Client programming

```
 use IO::Socket;
$socket = new IO::Socket::INET (
                PeerAddr  => '127.0.0.1',
                 PeerPort  =>  7008,
                  Proto => 'tcp',
                      )
or die "Couldn't connect to Server\n";
   $socket->recv($recv_data,1024);
   if($recv_data){
   print "Sum is is $recv_data\n";
   }
   else
   {print("Server is not working:Restart the sever and recompile the server program\n");}
   sleep(20);
```

Server Programming

```
use IO::Socket;
$| = 1;
$socket = new IO::Socket::INET (
                      LocalHost => '127.0.0.1',
                       LocalPort => '7008',
                        Proto => 'tcp',
                         Listen => 5,
                         Reuse => 1
                       );
die "Coudn't open socket" unless $socket;
print "\nTCPServer Waiting for client on port 7008";
while(1)
{
        my($new_sock,$buf);
     $buf=sum2();
        $client_socket = "";
        $client_socket = $socket->accept();
        $peer_address = $client_socket->peerhost();
        $peer_port = $client_socket->peerport();
        print "\n I got a connection from ( $peer_address , $peer_port ) ";
     $client_socket->send($buf);
     close $client_socket;
sub sum2() {return 7+3;}
}
```

V. Python Programming

Python is preprocessor and portable language. Python has import and other features for green computing

a) Recursion

```
# factorial
def fact(n):
   if (n <= 1):
```

```
    return 1
  else:
    return n * fact(n - 1)
```

b) Regular

```
#Searching text
txt = "Artificial Intelligence"
x = re.search("Intel", txt)
print(x)
```

c) Object-Oriented

```
print(fact(6))class table():
  # init method or constructor
  def __init__(self, customer, barrar, table):
    self.cust = customer
    self.barrar = barrar
    self.table = table
  def show(self):
    print("customer is", self.cust )
    print("supplier is", self.barrar )
    print("table  is", self.table )
# both objects have different self which
# contain their attributes
table1 = table("rama", "barrar1", "table1")
table2 = table("krishna", "barrar2", "table2")
table1.show()
table2.show()
```

The output is given by

```
customer is rama
supplier is barrar1
table  is table1
customer is krishna
supplier is barrar2
table  is table2
```

Python is portable using import.

```
import string
text = input('text: ')
symptoms = text.split()
symptom1='appreciating-colors'
symptom2='glaring'
symptom3='recognizing-faces'
```

If symptom1 in symptoms and symptom2 in symptoms and symptom3 in symptoms:
Print ('patient diagnosed cateract')

d) Client and Server

Python is portable to use for client/server programming.
For instace,

Server

```
import socket
serv = socket. Socket (socket.AF_INET, socket. SOCK_STREAM)
serv.bind(('0.0.0.0', 8080))
serv.listen(5)
while True:
  conn, addr = serv.accept()
  from_client = ''
  while True:
    data = conn.recv(4096)
    if not data: break
    from_client += data
    print from_client
    conn.send("I am SERVER<br>")
  conn.close()
  print 'client disconnected'
```

Client

```
import socket
client=socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
client.connect(('0.0.0.0', 8080))
client.send("I am CLIENT<br>")
from_server = client.recv(4096)
client.close()
print from_server
```

VI. Green Computing Technology

Green computing technology mainly has two criterions fundamentals of computer science and nature of computer science.

a) Fundamentals of Computer Science

Fundamentals of computer science may be defined as

Finite Automata
Regular Expression
Context-Free Grammar
Turing Machine
Digital Logic

b) Nature of Computer Science

Some of Nature of computer science may be defined as

Nature of Clouds
Nature of Neurons
Nature of Genetics
Nature of Trees and Forest
Nature of Proteins

VII. Conclusion

Perl and Python are best for Green Computing or clean computing. Perl is regular language and powerful at sever side programming. Python is pre-processor and it is portable with import feature. We try to discuss m Perl and Python programming languages for green computing.

References Références Referencias

1. Wojtek Kozaczynski and Grady Booch, "Component-Based Software Engineering", IEEE Software, 1998, pp.34-36.

2. Alan W. Brown and Kurt C. Wallnau, "The current state of CBSE", IEEE Softwarepp.3 pp.7-36, 1998.
3. Elaine Weyuker, "Testing Component-Based Software: A cautionary Tale", IEEE Software, 1998, pp.54-59.
4. Tom Digre, "Business Object Component Architecture", IEEE Software, pp.60-69, 1998.
5. Pamela Zave and Michael Jackson," A Component-Based Approach to Telecommunication Software", IEEE Software, 1988, pp.70-78.
6. Israel Ben-Shaul, James W. Gish, and William Robinson, "An Integrated Network Component Architecture", IEEE Software1998,, pp.79-87,1998.
7. Szyperski, C., Component software: Beyond Object-Oriented Programming, Addision Wesley Longman, and Reading, Mass., 1998.
8. Cox, B.J., Object Oriented Programming: An Evolutionary Approach, Addison Wesley Longman, and Reading, Mass., 1987.
9. Rumbaugh, J, Blaha, M, Premerlani, W, Eddy, F, Lorensen, W, Object- Oriented Modeling and Design, Prentice-Hall, NJ, 1991.
10. Booch, G, Object-Oriented Analysis and Design with Applications, Second Edition, Benjamin/Cummings, Redwood city,CA,1994.
11. Booch, G., Roumbaugh, J. and Jacobson, I., The Unified Modeling Language-Use Guide, Addison-Wesley Longman mc, Reading, MA, 1999.
12. P. Venkata Subba Reddy, "Object-Oriented Software Engineering through Java and Perl", CiiT International Journal of Software Engineering and Technology, vol.5, 2010, pp.29-31.