EasyChair Preprint

№ 3258

Green Computing with Regular Language Perl

Venkata Subba Reddy Poli

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

April 25, 2020

Green Computing with Regular Language Perl

P.oli VenkataA Subba Reddy
Department of Computer Science and Engineering,
College Aof Engineering,
Sri Venkateswara University,
Tirupati-517502, India.
e-mail:vsrpoli@hotmail.com

*Abstract-*Perl is regular language. Perl is powerful in server side programming. The software is made faster by using Perl It is powerful for POST method for Web programming particularly serves side programming. It is regular (RE) language, Apart from all the features, it portable and green programming language. The advantage of Perl is it can be used as object-oriented (OO) or non object-oriented (Non O-O) programming language.

Keywords— regular expressions object oriented, client/server, parallel and distributed, Perl

1. Introduction

Software Engineering is mainly facing challenging with Project Development. Software Engineers are going with Object-Oriented for Project Development. Still they are needed for better computing i.e., Green Computing. The Green Computing is the Design and Programming skills to develop the software projects. There are different stages algorithms level, design and analysis level and programming level for Green Computing. For instance recursion, parallelism, time complexity at algorithm level ,Object-oriented at analysis and design level and regular expression at programming level.

Green Computing or Clean Computing is portability and performance of Programming to solve different type of problems like Scientific, Business, Engineering, Medical. The Green Computing is the Programming skill in which the Program has portability of the code, suitability of the problem and less computation time. There are different techniques for Green Computing like recursion, parallelism, regular expression and Object-oriented. The recursion is calling function itself. The parallelism is computing the number of risks at a time. The regular expression is simplifying number of instructions. The Object-Oriented is module of program is independent

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.

Programming languages are designed based on Automata. Context-Free Language is recursive representation of Finite Automata. For instance C, PL, Pascal is designed based on FA.

The regular language is designed based on regular expressing. Regular expression is simple representation of Finite Automata. For instance Perl is designed based on RE.

The advantages of Perl are
Perl is used as General purpose language
Perl is used as Regular Language
Perl is used as Object-Oriented language.

The language is intended to be practical (easy to use, efficient, and complete)

Rather than beautiful (tiny, elegant, minimal).

It supports both procedural and object-oriented (OO) programming,

Green Computing may be studied at three stag of Algorithms

Analysis and Design
Programming

Algorithm is defined as step by step procedure to solve particular problem.

There are different methods. Mainly consist of
Brute force
Divide and Conquer
Decrease and Conquer
Transfer and Conquer
Dynamic
Sequential and Binary
Depth-First, Depth-First and Heuristic
Backtracking
Iteration
Recursive
Time Complexity
Space Complexity

Some of the above methods are used to design Algorithms.

For example to design the algorithm for particular problem, the Iterative or Recursive may be used. For green computing, Recursive Algorithms is used. Similarly, the divide and conquer method may be used for green computing with the parallel computing.

Instead of designing conventional algorithms, the above methods are useful for green computing.

2 Analysia and Design

They are mainly tow analysis and design methods Non Object-Oriented or Conventional and Object-Oriented. The Object-Oriented technology is better. The Componentware Technology with Object-Oriented is more helpful for green computing

Some basics of Component ware Technology are discussed in the following.

Componentware Technology (CWT) is emerging discipline in application software development. Many researchers contributed to bring out this new Component Technology [1, 2, and 3], applications and implementation tools [3, 4, and 5]. The Organizations are expecting new technologies for their expansion of their business and activities, instead of going for new software for their needs. They are looking for existing system is to be reuse and add off-the-self software The Componentware technology provides the Organization needs in which the components shall be reuse and expandable, and incorporate other software systems.

Components: A component is independent, non-trivial and replaceable unit of a system. The component may be enterprise, Domain subsystem, domain objects and semantic primitives.

Software component: A software component is a package of one or more programs as a unit of specified interfaces and dependencies. A software component is deployed independently, substituted, reuses and Composition the Components of the system. For example, Business component system is implementation of business system and the component such as Sales order processing execute independently.

Component Analysis Technology: CAT is defined as abstract components. These abstract components require technologies to define components, relationships, interfaces, actions, security and distribution. There is no need to define new techniques for the analysis. The existing Object-Oriented Analysis techniques CAT such as OMT of Rumbaugh [9] and Rose of Booch [10] technologies shall be extended.

Component Design Technologies: CDT is defined as system architecture that understands by CAT. This will concentrate on well-defined architecture and configuration of components. The Components are deployed, maintained and evaluated continuously. Multiple configurations will also provide similar configurations. The existing Object-Oriented Design technologies CDT such as OMG's UML [11] shall be extended

Composition of the Components: Composition of Components may be defined as Composition of nested types, Composition of Components are subsystems and composed components from semantic primitives such as rules and relationships.

The Component model consists of mainly components are connected to the infrastructure and making references to the other components, defining messages between components, and multiple interactions among components to provide secure communications between components. The components are extracted from entire enterprise. Semantics primitives of enterprise, interfaces, communication among components, threading and distribution must share the infrastructure. The examples of connectional distributing infrastructure are such as DCOM, OMG's Corba. Enterprise Java Beans and OMG's Corba are uses to define Component models and Structured domain specific concepts. Feature trend is the integration of Microsoft COM+ and DNA and Sun's J1NI for component ware for application software development.

The Component model is defined by Communication among Components and interfaces for use in distributing Computing Infrastructure. The Component Based Development is based on the Object-Oriented Technologies such as Object- Oriented Analysis Technology (OOAT) and Object-Oriented Design Technology (OODT). The Component model consists of Component semantic model and Component Architecture. Component semantic model describes domain concepts. Sometimes it may refer to component conceptual model. The component model consists of two levels component semantic model and component architecture.

2.1 Characteristics of Components

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 externally accessible view.

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

As Component software extended, the components are extendable.

The component must be relocate and replace a component for other implantations or 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 in to the other systems. Sometimes this maybe referred as off-the-components.

2.2 Component Semantic Model

Component semantic model has set of semantic concepts that support component technology. Some of the concepts are attribute changes and specifications.

Attribute changes: Notification mechanism will perform attribute changes in which the changes are only imposed within the components.

Specifications: Specification is the structure relationship between the components. The operations must incorporate in change-delete-create mechanisms during implementation.

The operations are such as contrast, domain specific concepts such as distributed and communication among the components. The following are characteristics of semantic model.

Object model: It includes interface and type definitions, abstract data types, and operations, accepts, and attributes, multiple inheritance and modules.

Structured semantics: These will specify the relationship between types within the component.

Component assembles semantics: These semantics are used to assemble components and generated into framework specific implementation.

Extendable semantic components: The components are extensible to multi versions of components and semantic concepts can be applied to domain architecture.

Life cycle partitioning: The partition defines explicitly behavioral and structured semantics with state model. The behavioral abstraction includes state transactions, event-condition- action rule. An event is observable change in what may happen to an object. Event triggers specify rules and dependencies. The components are produced and defined it from UML diagrams.

3 Component Architecture

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

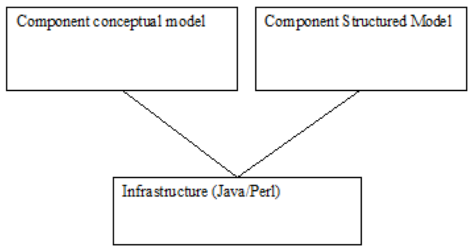
Component conceptual model
Component Structured Model
Infrastructure (Java/Perl)

Fig. 1 Component Architecture

3.1 Component Conceptual Model

This model mainly consists of two techniques OOA and OOD of the product how to components share and communicate and understanding of infrastructure. Some of the characteristics are, components are connected to the infrastructure, making references to other components via naming scheme, Guarantee of message transfer, transactions consisting of multiple interactions and providing security constraints. The components are defined using component technologies and these components are integrated and developed based on the component conceptual model.

3.2 Component

Infrastructure technologies provide the execution environment for of component semantic model and implement domain concepts with framework the main requirement of component of infrastructure is component interface by some component infrastructure. Components is implemented as functional that reflects role in the system and an extra functional.

The prominent component infrastructure is Enterprise Java Beans, Active X, Visual Basic Controls, Corba, DCOM, COM+ etc. The Microsoft with DCOM and SUN with Enterprise Java Beans, these two infrastructure technologies provide semantics of interfaces, communication among the components, threading and distribution. The integration of these technologies are emerging such, as Microsoft integration of COM+ANA and Sun's Jini will be used for further component of application software development.

3.3 Structure Domain Concepts

The Structure domain concepts are instantiated as framework-specific to implement the component solutions for problems. These concepts provide services and facilities using framework-specific conventions.

Frame-work: A framework provides services and facilities. It provides an execution environment for domain concepts implementation. It provides formal mechanism for defining interface such as OMG's interface language, Corba infrastructure transparencies incorporation and Enterprise Java beans. Universal Component Architecture (UCA): Universal Component Architecture is integration of independent Component Architectures(ICA) using interfaces. These interfaces provide distributed environment and remote access. The infrastructure technologies used for UCA implementation such as JavaRMI.

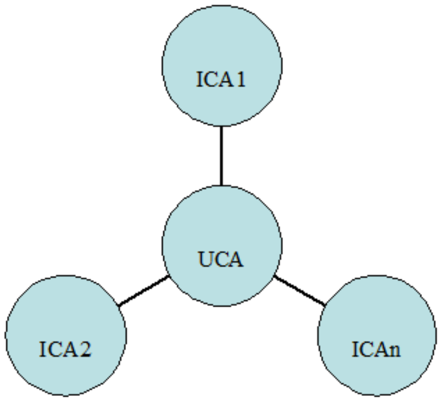

Fig. 2 Universal Component Architecture

3.4 Componentware Implementation

The component model is translated in to 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 ware developed from the scratch.

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

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

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

Domain system: It represents a particular are of components.

Domain object: It represents a particular process of components.

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

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

3.5 Componentware testing

The main advantages of components are independent, reusable and extendable to new developing software product. The testing is more important when reusing the components and off-the-self components. There are number of testing techniques described based on the source code availability. The software components which are developed for one project cannot be used to another project without proper checking.

Unit testing: The Unit testing is testing of individual testing. This is white-box testing using component's actual source code.

Integrated testing: Individual tested components are integrated and system is formed. This will be on interface code such as Java RML.

System testing: This will be done without reference to the code details.

4 Perl Programming

The Programming Languages fall under different paradigms Imperative, Functional, Logical, 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 $M=\{\Sigma, Q, \delta, q0, F\}$

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

The CFL is defined as $M= < V, T, P, S>$

The grammar $G= \{ A \rightarrow \alpha w\}$, where $\alpha \in V$, $w \in \{ N \cup \Sigma\}$

The regular grammar $G^*= \{ A \rightarrow \alpha w^*\}$, where $\alpha \in V$, $w^* \in \{ N \cup \Sigma^*\}$

For instance,

Σ=a-z(a-z, 0-9)

Id= num, id=x1 etc.

Σ^*={a-z, 0-9}*

Id=a-z(A-Z, 0-9)*

Id={x, x11, num, sum, sum12, …}

Programming Languages mainly four paradigms. Imperative, Regular, functional, Logical paradigms. C, Java, FORTRAN, Cobol etc., comes under Imperative.

Perl is for Regular, Lisp is for Functional and Prolog is for Logical.

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

Portable
Debugging
Reusability

The Computer Programming may be defined as combination of Data Types, Data Structure and Algorithm.
For instance
Data Types + Data Structures + Algorithm= C language
Data Types + Algorithm= java
Algorithm= Perl

Perl is only Regular Language. Perl can be used as Non Object-Oriented and Object-Oriented. Perl is Portable because Algorithms are directly defined in Program. Perl is mainly considered for green computing or green programming. Perl is also a general-purpose programming language

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

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

}

#Threads Acending and decending Order
use threads;

           $thr1= threads->new(\&ascending);
          $thr2= threads->new(\&decending);
                  ;
           sub ascending {
                my $num=0;
do { $num=$num+1;
 print " $num\n";
           }
            while ( $num<10)

}
sub decending {
              my $num=10;
do {  print " $num\n";
$num=$num-1;

           }
            while (  $num>0)
}
$thr1-> join;
$thr2->join;
#include <unistd.h>

       pid_t fork(void);

       if (fork())
      {
my $num=0;
do { $num=$num+1;
 print " $num\n";
}
       else
       {
                   my $num=10;
do {  print " $num\n";
$num=$num-1;
```

4.1 Perl as Regular Language

A regular expression is simply Expression of Finite Automate.
Consider the Regular Expression
num=digit*. digit)+
digit=0-9

matching

Regular expressions are used to match the pattern, sting with
"m//", "s///", "qr//" and "split" operators
Simple string matching

"Hello pvsr" =~ /pvsr/; # matches “pvsr” in Hello World”

The metacharacters are

{}[]()^$.?*+?\
/item[0-9]/; # matches 'item0' or ... or 'item9'

For instance
$a="Deer will eat food in food items"
$a =~ s/food/grass/;   # replaces food with grass in $a
$a =~ s/food/grass/g;   # it replaces all instances of food with grass in $a

4.2 Perl as Object-Oriented

The advantage of Perl is besides Regular, it is used to write programs in Object-Oriented and non Object-Oriented.

Perl supports Object Oriented programming. It simple and easy to write OO programs. OO Perl is based on Perl's concept of packages. OO program is a package in Perl. A method is simply a Subroutine.

1. An object is simply a reference that happens to know which class it
   belongs to.

2. A class is simply a package that happens to provide methods to deal
   with object references.

3. A method is simply a subroutine that expects an object reference
   (or a package name, for class methods) as the first argument.

A class is a package. An object is reference.
A class contains data and methods. An OO programming is set of classes and is called package in Perl.
different approach the with object experi-
ence knowing about subroutines , references
and packages

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;
```

Inheritance

Object-oriented programming systems all support some notion of inheritance. Perl has @ISA method.

Consider this class:

```
    package Employee;
/ use Person;
    @ISA = ("Person");
    1;
```

4.3. perl threds

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->jpin;
```

4.4, Perl fork

The fork is function shall create a new process from the existing process which is main process or parent process by defining as

```
    #include <unistd.h>

    pid_t fork(void);
```

The fork() crete chilp process with unique id.

for instance

```
    #include <unistd.h>

    pid_t fork(void);

    if (fork())
   {
child process
}
    else
```

```
        {
        parent processing
        }
```

for example

```
#include <unistd.h>

        pid_t fork(void);

        if (fork())
       {
       print " pvsr1\n";
}
        else
        {
        print "pvsr2\n";
```

This program sall print
pvsr2
pvsr1

4.2 Perl as Object-Oriented

The protocols are based on one-line messages and responses end with "\n" and in case of multi-line messages and responses that end with "\n.\n" terminates a message/response.

Internet Line Terminators

The Client-server communication that might extend to machines outside of your own system using Internet-domain sockets .

```
            #!/usr/bin/perl -w
            use strict;
            use Socket;
            my ($remote,$port, $iaddr, $paddr,
$proto, $line);

            $remote  = shift ?? 'localhost';
            $port    = shift ?? 2345;  # random port
            if ($port =~ /\D/) { $port =
getservbyname($port, 'tcp') }
            die "No port" unless $port;
            $iaddr   = inet_aton($remote)              ??
die "no host: $remote";
            $paddr   = sockaddr_in($port, $iaddr);
$proto   = getprotobyname('tcp');
            socket(SOCK, PF_INET,
SOCK_STREAM, $proto)  ?? die "socket: $!";
            connect(SOCK, $paddr)    ?? die
"connect: $!";
            while (defined($line = <SOCK>)) {
               print $line;
            }

            close (SOCK)           ?? die "close: $!";
            exit;
```

The kernel shall choose the appropriate interface on multihomed hosts for address

5 Conclusion

Green Computing is defined as portability in design and coding. Component technology is new trend for software development. The organizations for their software product are expecting reuse and extendable whenever they have be*en expanding or restructuring the business. The component technology will provide domain application needs. The cost of the software product is reduced vastly whenever they go for further expansion or new developing system. The Green Computing with Componentware Technology and Perl programming will simplify the design and programming

REFERENCES

[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.