Knowledge-Based EXP Framework (Verifying System and Product Configurations Using MS-Excel, XML and PERL Script)

Sambit Kumar Patra, Krishna TG

Abstract*—Nowadays, as computers get more advanced each day, so do the software being written for them. This advancement brings about increasing complexity in system configurations. Most, if not all, requirements are mentioned in lengthy user/installation manuals, reading those becomes all the more a tedious job. Small scripts are being used for automation to verify the configuration. However managing the scripts became more tedious as the numbers of configurations become huge. We figured that it would be so much easier on everyone if there could be a framework in place to do the verifications, instead of someone doing it manually. We have come up with a knowledge-based EXP framework which can be extensible using Microsoft Excel, XML schema and PERL script which we intend to verify, things that overlooked, system health check up, product specific optimal value, database and kernel.

***Index Terms*— knowledge-based system, EXP framework, system health check-up tool, verifying configuration tool.**

I. INTRODUCTION

The signaling components that we work with at Comviva Technologies are integrated with our SMS and USSD platforms on-field. So for the implementation, engineers on-site, the manual system verification before installation of a product of ours takes up 2-3 days. The biggest challenge for an implementation engineer is to verify the system by checking requirements specified by both the signaling as well as the SMS/USSD team. The signaling components require several configurations to be scrutinized and also synced over various files. All these verifications have been automated using scripts but the validation of the results of each of the scripts proves to be a tedious task altogether.

Installation of a complete signaling product would typically include checks such as OS version, system architecture, values of environment variables, memory status, database configurations, file permissions and various kernel-level configurations. These features vary depending on the type of deployment and as such, could be overlooked if being done manually. Also, in the case of occurrence of any issues, a support engineer would have to manually validate a long list of these checks.

Manuscript received October 2013.

Sambit Kumar Patra (Senior Technical Lead), Mobile Financial Solutions, Mahindra Comviva, Bangalore, India.

Krishna TG, Core Engineering Group, Mahindra Comviva, Bangalore, India.

A major challenge that is faced in such kind of validation is that the list of checks is not static. The requirement checks for a product increase with time and with product up-gradation.

There are two ways of doing something; manual or automated. In the same way, verification of a computer system prior to the installation of particular software can either be done manually, or by running certain automated scripts. The manual verification would require a person to go through installation manuals provided by the software developer, and would demand a lot of patience and care.

Automated system configuration solves only part of the problem. In the course of its lifetime, a server will undergo numerous configuration changes, software and operating system patches, and security updates. The automated verification would require some scripts to be written beforehand, and a person could be told to match the result of each script with a list of possible options, and then infer what each of it means.

However, this type of automated checking would require some level of care to be taken, in order to ensure the results being interpreted correctly. Using our proposed framework, we intend to eliminate this possibility of error too. We have verified our algorithms by the writing scripts for hardware specification, OS, database, signaling stack, SMSR, USSD and various products across the organization.

II. PROBLEM DEFINITION

A. Challenge to Support and Implementation people:

Developers only help in forming the product. Implementation and support engineers are the ones who finally have to ensure that the product works efficiently in the field. The biggest challenge faced by the implementation and support groups is the fact that since they were not part of the development of a product, their scope of work is limited to the FAQs and READMEs shared by the product development teams. This, in turn, needs careful reading of the same, and more often than not, one misses out on minor details in these documents.

B. Automation with XML:

Shell and Perl scripts are provided as a means of automating debugging, as well as to gather information regarding a particular issue before it can be reported to the development team for fixing. The interpretation of the outputs of scripts is way more tedious and also important than simply executing them. So it made sense to have a framework which could be used to execute scripts, and give clear objective results for the same. So we thought of having an XML file to hold these scripts to be given as input to a Perl script which would in turn execute the scripts, interpret the results and tabulate the same in an HTML file for easy viewing.

C. Writing/Modifying XML for inexperience will be difficult:

However, editing XML files is complicated, and one needs expertise to do so.

III. MOTIVATION

A. Motivation on Rough set approach to knowledge-based decision support:

Software installations require several changes to a system, which would all probably be shipped along with the package. Software runs with a certain range of system configuration. We may categorize them as pass, critical and warning stage. For example, in critical condition, software cannot be installed with the available memory size and CPU speed. In warning condition, software may install with certain range of memory. Looking at these categories and ranges we follow rough set approach to knowledge-based decision support [1].

Information exchange processes are often characterized by the need of translating from one data format into another, in order to achieve compatibility between information systems. This framework consists of an MS-Excel sheet converted to a backend to XML format and then given as an input to a Perl script, which would then generate the report of the verification in a HTML format.

B. Motivation for Perl Script and XML as Back End:

Harlan Carvey[8] , defined a framework in Perl script to monitor windows security system. To avoid hard-coding threshold values in the script, he used XML file to store the threshold information. The XML script compares the static values with threshold values. The disadvantage of this algorithm is, for version to version every time we need to keep add and update the static values. We modified the algorithm to execute the commands dynamically, and compare the results with 'N' number of values.

C. Motivation for Excel as Front End:

A product contains a lot of configuration values. The system, OS and product versions keep on changing constantly.

Writing the huge XML scripts and editing it will be difficult at the time of development. To reduce this time, there is a need for a good editor of XML file. Once it has been written, it can be updated by Implementation and support team.

Import and export XML file from Excel is easy. Implementation and support engineers are well-versed with writing simple shell and Perl scripts, and also using MS-Excel. This was the main motivation behind the creation of this framework. If there are some further checks to be added to an existing list, one just needs to write a script for the same, objectify the possible results and then add an entry for the same in the Excel sheet. Import and export of XML file from Excel is very easy. This edited XML file can then be used in the framework to obtain the additional results too.

IV. SOLUTIONS

Using rough set approach, we categorize different modules like OS and products. We collected the functionality that need to be checked before and after installation of a package. We wrote scripts to verify that functionality and categorized the severity as pass, fail and warning. We designed the Excel sheet according to these data. The Excel sheet (Table: I) has been designed so as to contain the bash command to execute a script from a particular path, and then to hold "N" number of different comparisons to which the output of the script would be compared to, and the corresponding inference of each of the possible results. Refer the flow chart in Figure 1.

TABLE I. FORMAT EXCEL FILE (XSLM)

Attributes	***Details***
Key	Key Number: easy to search
Module	Module Name
Functionality	Function Name
Command	Shell/Perl Script for execution
Match-X	Rule
Value-X	Value
Severity-X	Status (Pass/Fail)
Description-X	Status in Details

This Excel sheet is then converted to an XML file and then given as input to the Perl framework, which then executes each of the commands mentioned in the XML file, and generates a human-readable report of the entire exercise. This idea of ours successfully solves our problem of encountering any human error while verifying the system/product configurations prior to use.

Illustrative Example:

In Table-II, we have defined few check lists to verify the OS module only. The functionality column defines the check type. The command column defines the check script to execute on the system. The "match" column contains the conditional operator to match. The "value" column containing the value is to be matched. The number of match cases can be increased depending on the requirements.

TABLE II. EXCEL SHEET (EXAMPLE)

Key	***Module***	***Functionality***	***Command***	***Match 1***	***Value 1***	***Severity 1***	***Description 1***	***Match 2***	***Vcalu 2***	***Ssevierty2***	***Description 2***
1	**OS**	free_disk_space	df -lk | awk '{if(/^[^]+$/){remember=$0}else{printremember $0}}'	eq	""	Fail	Free disk space is not there	"ne"	""	**Pass**	Free disk space is availablle
2	OS	min_free_kbytes	sysctl vm.min_free_kbytes | awk '{print $3}'	">"	32769	Fail	min_free_byte is greater than 32768	"<"	8192	Fail	min_free_byte is less than 8192

Key	*Module*	*Functionality*	*Command*	*Match 1*	*Value 1*	*Severity 1*	*Desription 1*	*Match 2*	*Vcalu 2*	*Sseverty2*	*Description 2*
3	OS	net.core.rmem_default	sysctl -a | grep net.core.rmem_default |awk '{print $3}'	" > "	1048576	Pass	net.core.rmem_default is greater than 1048576	" < "	1048576	Fail	net.core.rmem_default is less than 1048576
4	OS	NIC bonding checking in /etc/sysconfig/network-scripts/	ls /etc/sysconfig/network-scripts/ifcfg-bond* >/dev/null 2>&1; echo $?	" == "	0	Pass	NIC bonding file exists	" > "	0	Warning	No NIC bonding file found
5	OS	NIC bonding checking in /sys/class/net/	cat /sys/class/net/bonding_masters >/dev/null 2>&1; echo $?	" == "	0	Pass	NIC bonding exists	" > "	0	Warning	No NIC bonding
6	OS	Status of IPv6	[-f /proc/net/if_inet6] && echo 'IPv6 ready system' || echo 'No IPv6 support found'	" eq "	IPv6 ready system	Pass	System has IPv6 support	" eq "	No IPv6 support found	Fail	System does not have IPv6 support

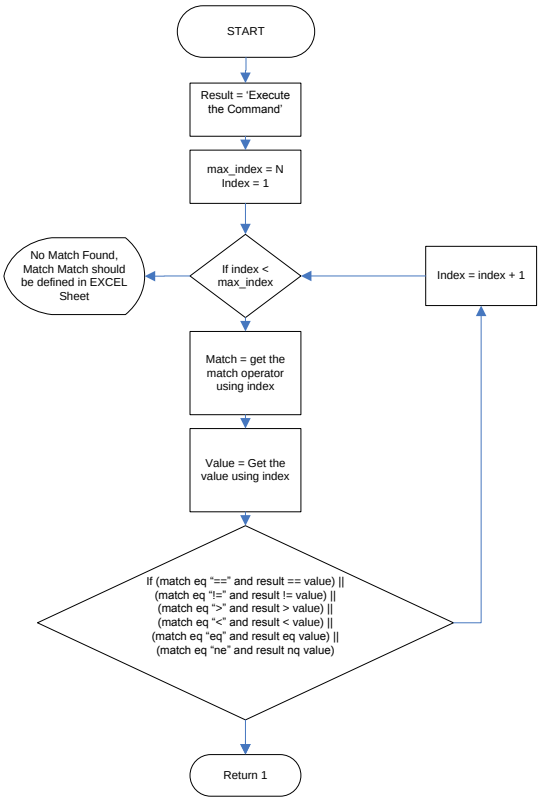

Fig. 1. Flowchart

In Figure-1, the Perl script executes the command in the command column. After execution of the command, the result will be compared to the value from the value column using the operator defined in the match column. If the match fails, the result will be compared to the next match operator with the value. When the result matches with the value it returns the corresponding description and severity.

```
use XML::Simple qw(:strict);
use Data::Dumper;
use File::Find;
use CGI;

open OUTPUT, ">$check_out_file" ;
$xml_in=$ARGV[0];
my $maxIndex = 2; #MAX INDEX IS EQUAL TO NUMBER OF MATCH CASES IN EXCEL SHEET
sub checkXML( $xml_in ) ;

sub checkXML( $xml_in )
{
   my $check_list = XMLin("$xml_in", forcearray => [ qw(key_val module var cond) ], keyattr => [] ) ;
   # Call HTML HEADER
   foreach my $row(@{$check_list->{Row}})
   {
      my ($matched, $result, $severity, $description) = getMatchedRule($row) ;
      if( $matched == 0){
         $severity = "Error";
         $description = "Command Error";
         $error++;
      }
```

V. ALGORITHM AND FLOWCHART

```
      # Call HTML FOOTER ($row, $result, $serverity,
$description) ;
   }
}

sub getMatchedRule($row)
{
   my $index, $match = -1 ;
   my $pass = $fail = $warning=0 ;
   my $result = `$row->{Command}`; #Execute the Script
   for ( $index = 1; $index <= $maxIndex ; $index++ )
   {
      my $match = $row->{"Match_"."$index"};
      if( $match )
      {
         my $val   = $row->{"Value_"."$index"};
         $severity = $row->{"Severity_"."$index"};
         $description   = $row->{"Description_"."$index"};
         $matched = is_rule_matched( $result, $match, $val);
         if($matched == 1)
         {
            last;
         }
      }
   }
   return ($match, $result, $severity, $description ) ;
}
sub is_rule_matched{
   my $result = shift ;
   my $match = shift ;
   my $val   = shift ;
   if(    ($match eq "'==" && "$result" == $val) ||
      ($match eq "'!=" && $result != $val)  ||
      ($match eq "'>" && $result > $val)  ||
      ($match eq "'<" && $result < $val ) ||
      ($match eq "'eq" && "$result" eq $val)|| ($match eq
"'ne" && "$result" ne $val)||
      ){
         return 1;
   }
   return 0;
}
```

Function checkXML(): It parses the XML and invokes the getMatchRule function. The function getMatchRule() executes the scripts , as per the number of index defined it calls the is_rule_matched() until gets a suitable match.

VI. RELATED WORK

Many applications have been developed with Microsoft Excel. Authors in [2] developed and presented a software called *BestKeeper* that determines the best suited standards. Excel is used for many decision support systems [3].

Today XML has reached a wide acceptance as intermediate representation or data exchange format [4, 5]. Authors in [6] present some strengths and weaknesses of XML technology. In System CXML [7], authors provide a lightweight and Open Source solution for source-to-source translations, structural information extraction, model visualization and documentation using intermediate XML data format. In [8], authors present an experimental evaluation of the latency performance of several implementations of Simple Object Access Protocol (SOAP) operating over HTTP, and compare these results with the performance of JavaRMI, CORBA, HTTP, and with the TCP setup time. SOAP is an XML based protocol that supports RPC and message semantics.

Author "Harlan Carvey"[9], proposed a solution to Monitor Windows security, CPU utilization. Here, authors store the configuration in XML documents and evaluate it using Perl Script. However the paper does not describe a unique solution to generate the XML configuration file. Authors in [10,11,12,13] purposed different XML editors.

VII. CONCLUSION AND FUTURE WORK

It is a widely-experienced problem in the world that we intend to do away with. Software installations require several changes to a system, which would all probably be shipped along with the package. A blind execution of an update script might alter certain changes to your system that are unwanted. So, using this framework, a pre-installation check could be performed to identify the required changes to be made. And most importantly, the time required for this type of activity has been brought down from 2-3 days to around 2-3 minutes. Later, this same framework could be used to verify if all the requirements are being fulfilled, prior to actual deployment of the software.

This framework in future can be enhanced so as to run as a cron-job. This would ensure constant monitoring of configurations. If any monitored parameter is modified, this framework could be used to alert appropriate personnel via an SMS or an e-mail.

ACKNOWLEDGMENT

We are thankful to Comviva Technologies, Bangalore, India, and in particular to Vikaas BV, Zunder L, Santosh BR, Vishwanath L Hugar, Prashant Aski and the entire Signaling, Performance, Support, Implementation, SMSR and USSD teams of Comviva for providing the resources to carry out this research work successfully.

REFERENCES

[1] Z Pawlak, Rough set approach to knowledge-based decision support, European journal of operational research, 1997, Volume 99, Issue 1, 16 May 1997, Pages 48–57

[2] Pfaffl MW, Tichopad A, Prgomet C, Neuvians TP (2004) Determination of stable housekeeping genes, differentially regulated target genes and sample integrity: BestKeeper—Excel-based tool using pair-wise correlations. Biotechnol Lett 26: 509–515

[3] Yunbo Lia, Qiping Shen, "Design of spatial decision support systems for property professionals using MapObjects and Excel".

[4] W3C, Extensible Markup Language (XML) 1.0, W3C Rec., Feb. 10, 1998.

[5] W3C, XML Schema Part 1: Structures, W3C Rec., May 2, 2001.

[6] D. Berner, J.-P. Talpin, H. D. Patel, D. Mathaikutty, and S. K. Shukla. Systemcxml: An exstensible systemc front end using xml. In FDL, pages 405–409. ECSI, 2005.

[7] David Berner, Jean-Pierre Talpin, "SystemCXML: An Extensible SystemC Front End Using XML"

[8] Dan Davis y and Manish Parashar, "Latency Performance of SOAP Implementations", Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium.

[9] Harlan Carvey, Jeremy Faircloth, Perl Scripting for Windows Security: Live Response, Forensic Analysis, and Monitoring

[10] Y. S. Kuo, Jaspher Wang, and N. C. Shih, "Handling Syntactic Constraints in a DTD-Compliant XML Editor"

[11] Y. S. Kuo , Lendle Tseng , Hsun-Cheng Hu , N. C. Shih, An XML interaction service for workflow applications, Proceedings of the 2006 ACM symposium on Document engineering, October 10-13, 2006, Amsterdam, The Netherlands

[12] Y. S. Kuo , N. C. Shih , Lendle Tseng , Hsun-Cheng Hu, Generating form-based user interfaces for XML vocabularies, Proceedings of the

2005 ACM symposium on Document engineering, November 02-04, 2005, Bristol, United Kingdom

[13] Marc Dymetman, Chart-parsing techniques and the prediction of valid editing moves in structured document authoring, Proceedings of the 2004 ACM symposium on Document engineering, October 28-30, 2004, Milwaukee, Wisconsin, USA.

Sambit Kumar Patra is working as Senior Technical Lead in Mahindra Comviva, Bengaluru from 2004. He is pursuing his postgraduate academic degree in Sikha "O" Anusandhan University, Bhubaneswar. He has published many papers. The last paper "JavaScript Interpreter Using Non Recursive Abstract Syntax Tree Based Stack", has been published in American Journal of Applied Sciences, 10 (4): 403-413, 2013.

Krishna TG obtained his Bachelor's degree in Electronics and Communication from Manipal Institute of Technology, Manipal in July 2012. He has been working since July 2012 as a software engineer at Mahindra Comviva as part of the signaling team in Core Engineering Group.