STM JOURNALS

Journal of Advances in Shell Programming

Volume 1, Issue 3
www.stmjournals.com

An Introduction of Smart Self-learning Shell Programming Interface

Ankush Rai*

Department of Applied Science, CRIAD Laboratories, Smiriti Nagar, Bhilai, Chhattisgarh, India

Abstract

To manage computer systems, the command line interfaces are heavily in use by computer administrators. Therefore, such tasks require a prerequisite know-how of the programming interface. Often, such tasks desired for automation requires repetitive coding and heavy interdependencies on certain logical input-output co-relation. Due to this criticality it is required that such automation of system administration jobs are often supervised manually. This paper presents an intelligent programming interface to help in jobs of automating system administration by fast paced coding with adaptive level learning and simultaneously supervising the procedural execution in logically multi-relational basis.

Keywords: *Automation, Shell programming, intelligent interface*

****Author for Correspondence*** E-mail: ankushressci@gmail.com

INTRODUCTION

Past study had shown the command-line interface is among the high rise tool availability for system administrator for automating the management, scheduling and troubleshooting [1]. As, command line interface has several advantages offers the following advantage for dynamic work flow environment:

1. It allows automate the task, when performing the same procedures multiple times across the cluster of machines.
2. It reduced the downtime of errors, which was once accomplished manually.
3. It preserves the knowledgebase to remain organizational for easy readability and execution.
4. Finally, it allows easy sharing of knowledge and shell commands with the colleagues.

It has drawbacks, which have been highlighted in the increasing computing era. The cost of authoring shell scripts are often too high and requires way more programming knowledge. Additionally, the same old scripts have been handed over and over gained without fully knowing what it accomplishes in certain number of steps; which makes it a daunting task to optimize. Also, the time and effort invested in diagnosing failures for automation in scripting is prohibitive. This paper demonstrates a smart learning algorithm based shell programming interface to overcome such problems in practice. It is an evolving system which learns sets of procedures from human-generated examples, and refined its behavior based on multiple relations between inputs outputs with that of the execution time using a sequenced cascaded neural network. Though using version space algebra; few handful work is already been accomplished [2–4]. The system was based on user feedback, while ours is independent of it.

METHODOLOGY: IMPLEMENTATION OF SMART SHELL

The shell scripting jobs for a system administrator for bringing changes to a server usually involve the following usual procedures:

- Begin the process on the server to make a note of the port number it chooses.
- Run the suite for test and pass its port number as an argument.
- Use a process in the given listing to determine the id of the server's process.
- Dispatch the kill signal to the server using its process id.

Now, with the smart interface based on the proposed smart shell scripting algorithm (SSSA); the developer can use it to automatically, build a test harness for such repetitive task. From the developer we have the segmented process and labelled test conditions to begin the training of the proposed test suite; it is required to frame up an adaptive algorithm to code feature conditions and its process orientation or ordering. Therefore, we use the cascaded neural networks for the same. To form an associative pattern between the neighboring sets of process blocks with the varying condition through the cascaded coding of training sets which requires three vectors:

(a) Bit rate of transmitting the displacement vector field.
(b) Bit rate for sequencing connectomes of information tree.
(c) The error rate in learning.

Given the segmented process sets $P_{operations}$ is given as: (x_1, y_1), (x_2, y_2), …, $(x_l, y_l) \in P_{operations} \times [1,0]$. Let $v_i \epsilon V$ be the set of displacement vectors for each of the neighboring nodes n_i. Such nodes represent the positioning of the feature sets. Therefore, in order to sequence the process from the reconstructed sequencing scheme we use the following SSSA algorithm.

Algorithm: Smart Shell Scripting Algorithm (SSSA)
Input: (x_1, y_1), (x_2, y_2), …,$(x_i, y_j) \in P_{operations}$, where x is the labelled operations and y is the conditions.
Output: $P_{schedule}(i, j)$

Step 1: while $l \leq \min\left(\sum_{i=1}^{N} D(v_i, n_i)\right)$ //D is the associative vector

Step 2: Evaluate the feasible value of target nodes R_{target}:

$$R_{target} = \sum_{i=1}^{N} D(v_i, n_i)$$

Step 3: Create target vectors for feasible neighboring nodes:

Loop: *for* 1 to n_i

$$\sum_{i=1}^{N} R(v_i, n_i) \leq R_{target}$$

Step 4: Using Lagrange multiplier λ for the activation function of the cascaded neural network:

$$CC(i, j) = \sum_{i=1}^{m} \sum_{j=1}^{n} sgn(D(v_i, n_i) + \lambda R(v_i, n_i))$$

end for loop
end while loop
Step 5: END PROCESS

The playback loop for redundant task continues on the next steps in the procedure, running the described SSSA test suite and bringing up a process listing shown in the Figure 1 below. The command for the kill operation requisite in the subsequent steps requires as its no argument, as the process id of the sequence of states previously generated in SSA for a currently-running server, which was printed out as a result of the process command is executed primitively. In this case, the proposed smart shell guesses about what the user required to run the command, using the labelled correct process id from the training sets. Though it has shown some major decision capability despite the fact that; when we had recorded the procedure and made the user to kill 675 process though it killed 20002 unnecessary process to make system fluent in port-port communication.

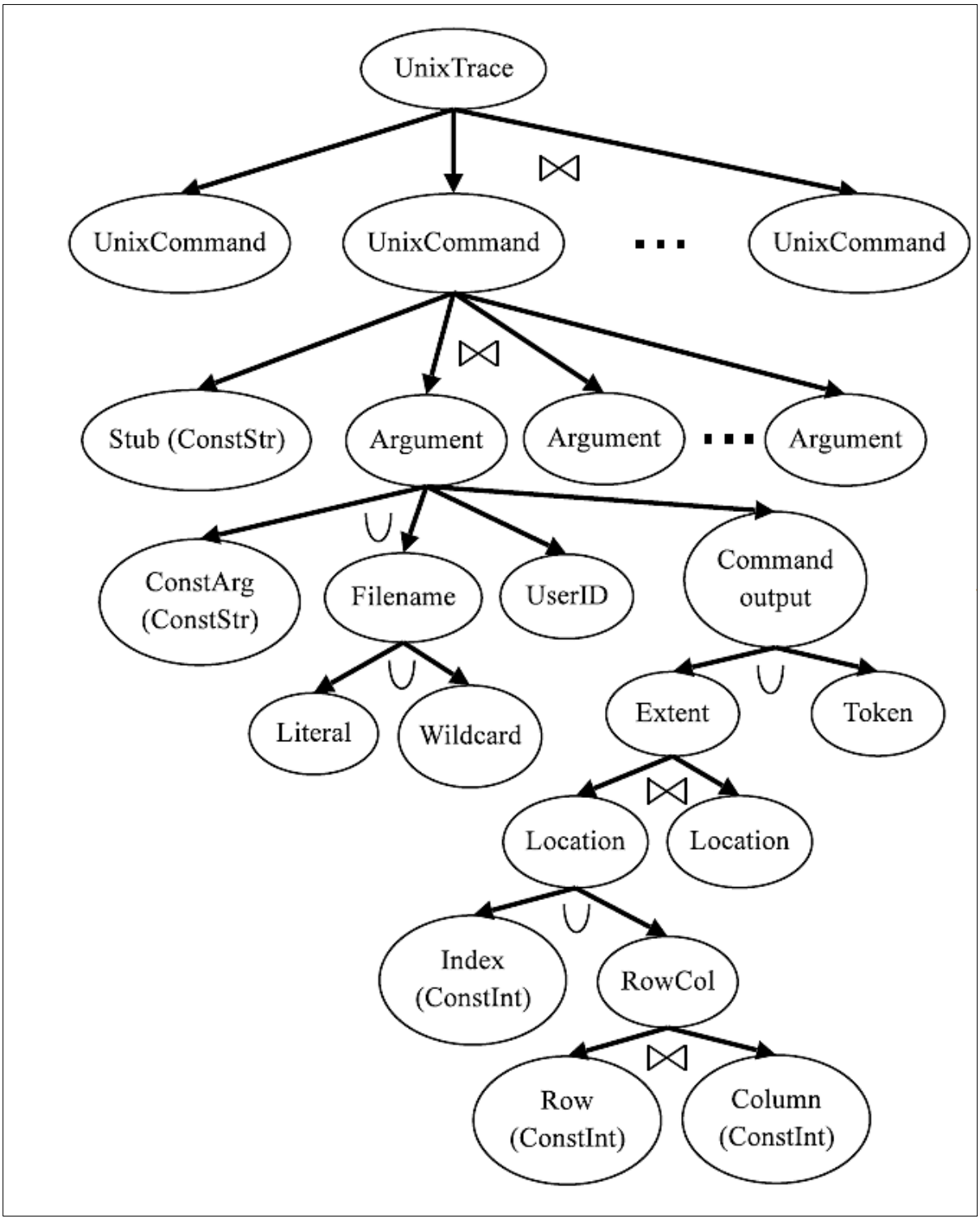

Fig. 1: *SSSA Version Space Depicting where the Bowtie Symbol Represents Adjoining of the Version Space whereas the Union Symbol Indicates a Version Space Union.*

CONCLUSION

Our work with the implementation of smart shell algorithm to automate a system has brought us near to formulate several desiderata for artificial intelligence as Autonomous Data Processors; which incorporates and assembles the human input and enable automation or job scheduling which requires no user to take over

the control at certain specific critical points during the execution of the learned process. In summary, we have showed an implementation of self-learning scripts.

REFERENCES

1. Kandogan E., Maglio P. P. Why don't You Trust Me Anymore? Or the Role of Trust in Trouble Shooting Activities of System Administrators. *In CHI 2003 Workshop: System Administrators are Users*.
2. Lau T., Wolfman S. A., Domingos P., *et al*. Programming by Demonstration using Version Spacealgebra. *Mach Learn* 2003; 53(1-2): 111–156p.
3. Lau T., Domingos P., Weld D. S. Version Space Algebra and its Application to Programming by Demonstration. *In Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, 527–534p.
4. Mitchell T. Generalization as Search. *Artif Intell* 1982; 18: 203–226p.