August 16, 2017

Smart State Management

Andy Lou
Hewlett Packard Enterprise

I-Chun Shih
Hewlett Packard Enterprise

Johnny Hung
Hewlett Packard Enterprise

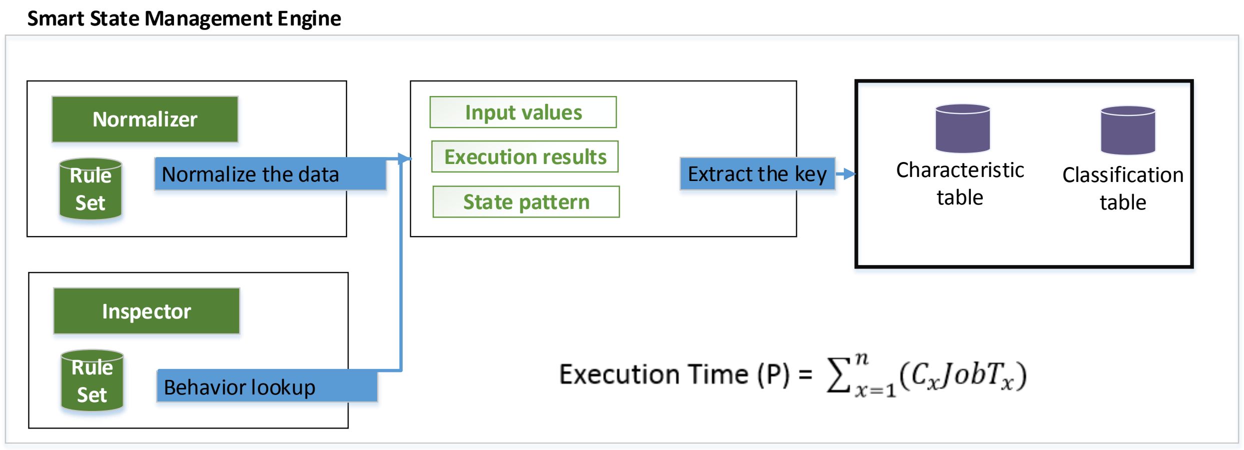
Ricky Tang
Hewlett Packard Enterprise

Smart State Management

A method of smart state management to make a software stateful is disclosed. This method involves determining where to define states within a software using a classification learning mechanism.

This mechanism includes a normalizer, which can extract the characteristics of a job and an inspector which can determine whether a state will be bypassed according to the characteristics or human-defined rules. The characteristics is a set of attributes containing input values, execution results, and pattern of states that are used by the inspector to determine whether a particular state within a future job is exactly part of this very characteristics.

In order to make the original state machine able to adapt a new stateful flow without changing the original flow or procedures. The smart state management engine needs to define entry points and exit points in different states of software execution.



When software execution starts, the inspector will look up the classification table and see if a particular state matches an entry in the classification table with the same state pattern. If the state does not match any entry in the classification table, the job will be executed, and then the normalizer will update both of the characteristic table and the classification table after the execution is complete. If the state matches an entry in the classification table, this state will be bypassed and proceed to the next state in the state pattern.

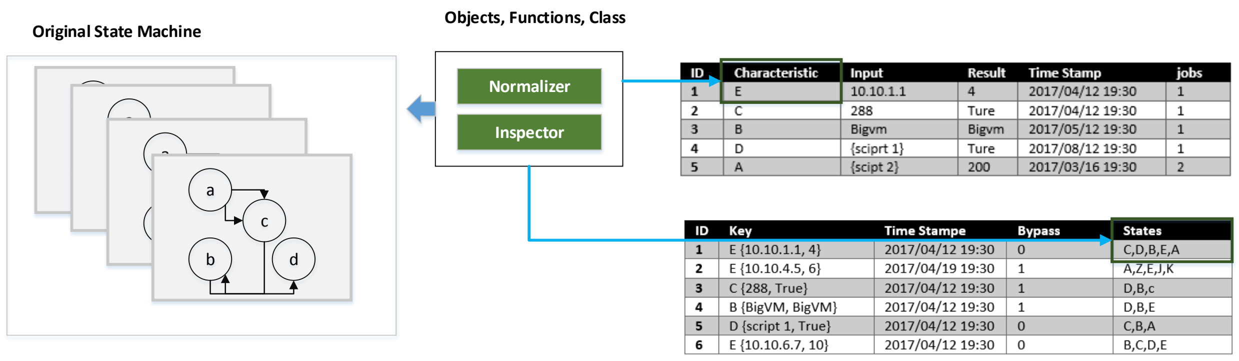
Characteristic table:

ID	Characteristic	Input	Result	Time Stamp	Jobs
1	E	10.10.1.1	4	2017/04/12 19:30	1
2	C	288	Ture	2017/04/12 19:30	1
3	B	Bigvm	Bigvm	2017/05/12 19:30	1
4	D	{sciprt 1}	Ture	2017/08/12 19:30	1
5	A	{scipt 2}	200	2017/03/16 19:30	2

Classfication table:

ID	Key	Time Stampe	Bypass	States
1	E {10.10.1.1, 4}	2017/04/12 19:30	0	C,D,B,E,A
2	E {10.10.4.5, 6}	2017/04/19 19:30	1	A,Z,E,J,K
3	C {288, True}	2017/04/12 19:30	1	D,B,c
4	B {BigVM, BigVM}	2017/04/12 19:30	1	D,B,E
5	D {script 1, True}	2017/04/12 19:30	0	C,B,A
6	E {10.10.6.7, 10}	2017/04/12 19:30	0	B,C,D,E

The way to make software execution stateful is mocking up the processes to predict what processes will be executed in a run and can be defined as a state. After the states are defined automatically by the normalizer and the inspector, these states can be used to record the state patterns and update the classification table.



ID	Characteristic	Input	Result	Time Stamp	jobs
1	E	10.10.1.1	4	2017/04/12 19:30	1
2	C	288	Ture	2017/04/12 19:30	1
3	B	Bigvm	Bigvm	2017/05/12 19:30	1
4	D	{sciprt 1}	Ture	2017/08/12 19:30	1
5	A	{scipt 2}	200	2017/03/16 19:30	2

ID	Key	Time Stampe	Bypass	States
1	E {10.10.1.1, 4}	2017/04/12 19:30	0	C,D,B,E,A
2	E {10.10.4.5, 6}	2017/04/19 19:30	1	A,Z,E,J,K
3	C {288, True}	2017/04/12 19:30	1	D,B,c
4	B {BigVM, BigVM}	2017/04/12 19:30	1	D,B,E
5	D {script 1, True}	2017/04/12 19:30	0	C,B,A
6	E {10.10.6.7, 10}	2017/04/12 19:30	0	B,C,D,E

Here is the expeted result:

Optimized Execution Time (P) = $\sum_{x=1}^{n}(C_x Job T_x)$

The C_x is the Bypass parameter (which can be 0 or 1) controlling either to bypass or to execute Job_x. Therefore, the more C_x values are applied to the formula, the more stateful the software execution can be. Furthermore, if more C_x values equal to 0, the fewer states will be executed and the more time can be saved.

Disclosed by Andy Lou, I-Chun Shih, Johnny Hung and Ricky Tang, Hewlett Packard Enterprise