Universidad de Buenos Aires

Facultad de Ingeniería

Año 2018 - 1° Cuatrimestre

75.10 Técnicas de diseño
Validador de reglas

Fecha: 11/04/18 Primera entrega

INTEGRANTES

Apellido y Nombre	**Padron**	**Mail**
Perez Bustos Drew, Maria Victoria	92060	vickyperezbustos@gmail.com
Galván, Pedro Nahuel	91977	nahuelgalvan@gmail.com
Prado, Maria Florencia	96626	pradomflorencia@gmail.com

Diseño

Mediante un parser se descompuso las reglas (counters y signals) con el fin de un manejo más amigable de las mismas.
Se decidió que el parámetro "state", el parámetro que devuelve la función initialize-process, sería un arreglo con cuatro diccionarios. Este va a mantener el mismo formato pero con los valores actualizados cada vez que pase por un process data.

State = [counters rules signals data-past].

- En counters guardamos los contadores de cada regla en un diccionario de la forma: {"name-rule" value}. En el caso que el contador tenga parámetros el valor va a ser un diccionario con los valores de parámetros de clave y el contador de valor, para este caso toma la forma: {"name-rule" 2 "name-rule2" { [true true] 1 } }.
- El segundo parámetro, rules, es otro diccionario de la forma: {:name-rule [[parámetros] condición]}, donde parametros es un vector con los parámetros de la regla, y condiciones la condición de la misma.
- Signals, análogamente a "rules" , es un diccionario de la forma: {:name-signal [resultado condición]}, donde el resultado y la condición corresponden a la mismas que entra en las rules para ese signal.
- Data-past, por último es un diccionario que tiene como clave la clave del dato y como valor una lista de valores que llegaron de ese dato. Tiene la forma: {:data-key: (data-value1 data-value2) }

En el caso de signal, que devuelve el process data, va a ser una lista con 1 diccionario que va a tener como claves los nombres de los signas y como valor el resultado de cada uno. En el caso que no haya datos para calcular el resultado no va a estar incluido el signal dentro del diccionario.

Signal = ({ nombre-signal1 resultado }).

Si no hay ningún signal para el cual se pueda calcular un resultado, el SIGNAL que devuelva el process-data va a ser del formato []. Esto lo hicimos de esta forma ya que los tests pedían que en el caso que haya alguno devuelva un listado con un diccionario adentro y para el caso que no haya que la cantidad de elementos del SGN devuelva 0 y si dejábamos el ({}) vacío el count (cantidad de elementos de la colección) va a devolver 1.

Hipotesis

- Tomamos como funciones posibles que nos entren como parámetros solamente las que están descritas en el enunciado del TP, es decir:
  =, != (para todo valor), OR, AND, NOT (booleanos), +, -, *, /, mod, <, >, <=, >= (para numericos) y includes?, starts-with?, ends-with? y concat (para string).
  También recibimos como válido el valor de "past", "current" y "counter-value".
- En el caso que entre un dato del cual solo tenemos un parámetro, guarda en el counter ese dato pero con el valor del parámetro faltante en nulo. Ej: si recibimos como dato {"spam" true} el spam-important-counter va a sumar en [true nil] uno.
- En el caso que no haya datos para calcular el resultado del signal, el mismo no va a estar incluido dentro del diccionario SIGNAL.
- Si no existe signal para el cual se pueda calcular un resultado, el SIGNAL que devuelva el process-data va a ser del formato [].

Ventajas

- Tenemos todos los datos a mano para cada evaluación que debemos hacer.
- Al aceptar solamente determinadas funciones, nos basta solo con convertir en string la función recibida y aplicarla.

Desventajas

- Cuando se reciba mucha cantidad de información, no va a resultar muy eficiente el hecho de recorrer todas las listas y diccionarios.
- Tener que guardar toda la información va a generar que se ocupe más espacio al entrar volúmenes más grande de información.

Esquema de la función que evalúa las condiciones y expresiones