Especificación en RSL de Componentes Basadas en Streams

Daniel Riesco; Berón, Mario; Montejano, Germán
Departamento de Informática
Universidad Nacional de San Luis – Argentina

e-mail: {driesco|mberon|gmonte}@unsl.edu.ar

Dosch, Walter
Institute of Software Technology and Programing Languages
Medical University of Lübeck, Germany
e-mail: dosch@isp.mu-luebeck.de

Resumen

Un stream es una secuencia finita o infinita de mensajes transmitidos sobre un canal. Los streams tienen propiedades las cuales hacen posible modelar software en dominios específicos. RAISE es un método riguroso para el desarrollo de software. RSL, es el lenguaje de especificación formal usado por el método RAISE. Se han desarrollado herramientas que permiten verificar automáticamente las especificaciones de los módulos de software construidos. Aquí radica la ventaja del uso del método RAISE.
En esta línea de investigación se estudia el formalismo de los streams y las componentes de software basadas en ellos. Se presentan aquí las especificaciones en RSL de las propiedades básicas de streams y de la componente scan, la cual recorre una secuencia de datos en tiempo serial aplicando una operación binaria, esta operación binaria es subespecificada para permitir su reuso. Junto con la componente scan se muestran sus aplicaciones en la construcción de nuevas componentes.

Palabras Claves: RAISE, RSL, streams, componente, especificación, esquema.

1. Introducción

Las componentes de software pueden ser especificadas por el formalismo de streams [4,5,6,7,8] que permite el estudio de las componentes de software basadas en las historias de entradas y salida de la misma.

La idea de las especificaciones de componentes, con el formalismo de streams, consiste en el reuso de componentes existentes, es decir la construcción de una nueva se lleva a cabo usando las definidas previamente.

Si bien la formalización de componentes de software usando stream brinda otra visión del estudio de las mismas, carece de herramientas automáticas que permitan chequear la correctitud de las especificaciones hechas usando éste formalismo. Es aquí donde el uso de método RAISE[1,2,3] (Rigorous Approach to Industrial Software Engineering) junto con sus herramientas automáticas juegan un rol central, ya que esta basado sobre los principios de desarrollo separado, desarrollo paso a paso, inventar, verificar y rigurosidad.

El punto importante usado en este trabajo es el concepto de desarrollo separado. Si se desea desarrollar sistemas de gran tamaño debemos ser capaces de descomponer su descripción en componentes y componer el sistema a partir de éstas. Cada componente puede ser especificada por diferentes ingenieros, pero se presenta el problema que cada uno escribe una función que otro quiere usar o reusar. Es simple chequear el nombre de la función, sus parámetros, y su tipo de resultado, pero no se puede afirmar lo mismo de la semántica de la función. Los esquemas de RAISE son usados para definir la semántica de cada componente.

El lenguaje usado para las especificaciones es RSL (RAISE Specification Language). RSL permite a los ingenieros especificar esquemas usando el contexto y esquemas parametrizados. Si una

función definida no está en el esquema, esta función se busca en el contexto. Este se construye escribiendo el nombre del esquema al comienzo de la definición de uno nuevo.

Otra característica usada en los esquemas de especificación de streams es el esquema parametrizado. El uso de esquemas parametrizados indica los tipos que el esquema recibe como parámetro permitiendo así la generalización de esquemas. Así, se definen las componentes usando estas características. Esto permite el reuso de las componentes ya especificadas.

En las secciones siguientes se describe brevemente el formalismo de streams, las especificaciones en RSL y los pasos que se siguieron en la investigación.

2. Especificación en RSL de Componentes Basadas en Streams

Para realizar el estudio del formalismo de streams y llevar a cabo la especificación en RSL, se tomaron los siguientes aristas:

1. Especificación en RSL de los transformadores de streams primitivos.
2. Especificación en RSL de componentes basadas en streams existentes.
3. Creación de nuevas componentes basadas en streams con sus correspondientes especificaciones en RSL.

A continuación se describen las tareas realizadas en cada uno de los aspectos mencionados arriba.

1. Especificación de los Transformadores de Streams Primitivos

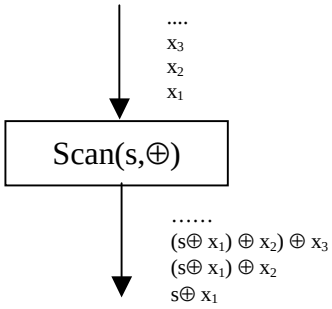
Los streams modelan una sucesión temporaria de mensajes sobre canales de una red de componentes de comunicación. Dado un alfabeto A, el conjunto A^* de streams finitos comprende todas las secuencias $A=<a_1,a_2,.....,a_k>$ de longitud $|A|=k>0$ con los elementos $a_i \in A$ ($i \in [1,k]$). Los streams finitos son generados desde el stream <> agregando elementos al frente del stream: $<a_1,a_2,.....,a_k>= a_1 \lhd a_2 \lhd ..... \lhd a_k \lhd <>$. La concatenación de dos streams finitos: $A= <a_1,a_2,.....,a_k>$ and $B= <b_1,b_2,.....,b_l>$. Produce el stream $A\&B =< a_1,a_2,.....,a_k, b_1,b_2,.....,b_l>$ en [5] se encuentra una síntesis completa de los operadores de streams. La especificación en RSL de estos operadores primitivos es directa porque los mismos vienen incorporados en el lenguaje. Los transformadores de streams usados en este formalismo se definen en base a los primitivos y siguen una lógica similar a la de la programación funcional y sus especificaciones pueden ser vistas en [9].

2. Especificación en RSL de Componentes basadas en Streams

Como se dijo en apartados anteriores el aporte que hace el formalismo de streams a la especificación de componentes de software es que permite el estudio de éstas basándose en las historias de entrada y salidas.

Se han especificado, usando el formalismo de streams, un conjunto de componentes sencillas que adquieren importancia por su generalidad, la cual se ve reflejada por su amplio uso en la creación de nuevas componentes, una de éstas es el scan. El scan es una componente dependiente del estado con un puerto de entrada y uno de salida. Ésta, consume un stream de entrada de datos y emite el stream de segmentos iniciales donde dos elementos son combinados bajo una operación diádica. El resultado de esta componente depende del valor del estado inicial y de la operación de combinación.

El comportamiento del scan puede visualizarse de la siguiente manera:



La especificación en RSL de esta componente puede ser vista en la figura 1.

```
scheme Function_Scan=
       class
       type
        B,
        A

       value
        scan: B x (BxA → B) → (A* → B*)
        scan(s,f)(A) is
          if A=<..> then
             <..>
             else
              <.f(s,hd(A)).> ^ scan(f(s,hd(A)),f)(tl(A))
          end
end
```

Figura 1: Especificación en RSL de la componente scan

Existen dos teoremas que adquieren importancia porque ayudan a la especificación de nuevas componentes, tal es el caso del Teorema de la Composición Paralela y el de Composición Serial.

El Teorema de la Composición Paralela afirma que:

$$(scan(s,⊕) \text{ x } scan(t,⊗))° \text{ syn} = unzip°scan((s,t), ⊕x⊗)$$

donde x denota la función producto y ° la composición de funciones.

El Teorema de la Composición Serial es el siguiente:

$$scan(s,⊕) \text{ x } scan(t,⊗)= map2(first) ° scan((s,t),Θ)$$

donde:

$$(x,y) Θ z = (x ⊕ (y ⊗ z), y ⊗ z)$$

El transformador de stream map2 aplica la función first a todos los elementos de una tupla de streams y el transformador first retorna el primer elemento de una tupla. Las especificaciones en RSL de estos teoremas pueden ser vistos en [9].

3. Especificación en RSL de Nuevas Componentes Basadas en Streams

En el apartado anterior se presentó la componente scan y se destacó su generalidad para especificar nuevas componentes, se presentó además la especificación en RSL de la misma y se delinearon los teoremas de la Composición Paralela y de la Composición Serial. En esta sección se

muestra la especificación con streams de una nueva componente: Un Contador Módulo N que usa al scan.

El contador consume un stream de números naturales con la siguiente regla: La entrada es acumulada mientras no ocurra un comando reset. Después de un comando reset el contador comienza nuevamente desde cero.

Como un primer paso se fija la interfaz sintáctica de la componente. Sea **N**={0,1,2,3,....} el conjunto de los números naturales . Para un número N $\geq$1, sea N={0,.....,N-1} los restos módulo N.

Un contador módulo se describe por el transformador de stream de orden superior

$$\text{COUNT: } N \rightarrow [(\mathbf{N} \cup \{\text{reset}\})^{\infty} \rightarrow N^{\infty}$$

Donde reset denota el comando reset. La componente combina el estado interno con la entrada bajo la operación $\oplus$: N x (**N** $\cup$ {reset}) $\rightarrow$ N validando:

$$s \oplus x = \begin{cases} 0 \text{ si } x = \text{reset} \\ s+x \bmod N \text{ si } x \in \mathbf{N} \end{cases}$$

El comportamiento de la entrada salida de un contador módulo esta dado por el transformador de stream:

$$\text{COUNT}(s) = \text{SCAN}(s, \oplus)$$

De esta manera se muestra como la componente scan puede ser usada para especificar nuevas componentes. Además del contador se han definido otras componentes que hacen uso del scan: Una celda de Memoria y un Concordador de Paréntesis, cuyas especificaciones pueden ser vistas en [9].

3 Conclusiones

El hecho de especificar en RSL, las funciones primitivas de streams, la componente scan junto con sus propiedades y sus aplicaciones, permitió observar que: la especificación de los operadores primitivos de stream es directa, la especificación de componentes se facilita por el reuso de las mismas y por su similitud con la programación funcional, el chequeo automático provisto por RSL es una gran ayuda para la correctitud de la especificación de la componente basada en stream.

Como un caso de estudio se esta analizando la posibilidad de especificar en RSL una arquitectura para la transmisión de videos en la web. Para esto se pretende en primera instancia realizar una especificación usando streams y luego traducirla a RSL permitiendo así el chequeo automático de las propiedades y comportamiento de las componentes.

Bibliografía

1. The RAISE Development Method. Chris, George; Haxthausen, Anne E.; Hughes, Steven; Milne, Robert; Perhn, Soren; Pedersen, Jan Storbank. Prentice Hall. 1995.
2. The RAISE Specification Language. Chris, George; Haxthausen, Anne E.; Hughes, Steven; Milne, Robert; Perhn, Soren; Pedersen, Jan Storbank. Prentice Hall. 1992.
3. RAISE tools. *www.iist.unu.edu*
4. The semantics of a Simple Language for Parallel Programing 471-475. Kahn, G.; Information Processing. J. L. Rosenfeld. Amsterdam: North-Holland. 1974.
5. A Survey of Stream Processing. Stephens, R.. Acta Informática 34, 491-594; 1997.
6. Scanning Stream. Dosch, Walter. Institute of Software and Programming Languages. Medical University o Lübeck. Proceeding of the ACIS. International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Application 61-67. Foz do Iguazu. 2002.
7. Views of a Bounded Stack. Dosch, Walter. Institute of Software and Programming Languages. Medical University o Lübeck. Proceeding of the ACIS. International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Application 11-17. Foz do Iguazu.2002.
8. Coroutines and Networks of Parallel Processes 993-998a. Information Processing; B. Gilchrist. Amsterdam: North-Holland. 1977
9. Formal Specification of Scanning Streams using RAISE. Dosch; W Riesco, D; Berón, M.. Argentina. 2003. Reporte Interno.