INTERPRETACIÓN DINÁMICA DE MÚLTIPLES LENGUAJES DE DOMINIO ESPECÍFICO

Héctor A. FLOREZ FERNANDEZ
Facultad Tecnológica, Universidad Distrital Francisco Jose de Caldas
haflorezf@udistrital.edu.co
Bogotá, Colombia

RESUMEN

El presente artículo, muestra el resultado de un proyecto que tiene como objetivo principal, la interpretación de diferentes DSLs (Domain Specific Language) los cuales pueden ser construidos con diferentes herramientas y tecnologías. Para lograrlo, se requiere una aplicación que permita controlar los diferentes DSLs creados, mediante reglas de ejecución preestablecidas. El caso de estudio adoptado para llevar a cabo esta propuesta, consiste en un juego en donde cada jugador proporciona su DSL y los componentes de las herramientas utilizadas para la implementación del DSL, así como los componentes necesarios para su interpretación. Esto indica que la aplicación que funciona como ambiente de ejecución de los DSLs, está en la capacidad de cargar en memoria dinámica todo el contenido que requiera proporcionar un jugador. Específicamente el juego consiste en un escenario de combate en donde puede haber hasta seis jugadores. Cada jugador conforma un equipo y cada equipo debe intentar destruir a los equipos rivales. Cada jugador puede tener en su equipo hasta cuatro tanques de guerra y cada tanque podrá ser interpretado por un DSL en particular. De esta forma, cada jugador puede diseñar cada tanque con un DSL diferente permitiendo incluir al escenario hasta cuatro DSLs por equipo. Cada DSL puede ser construido en diferentes lenguajes como ANTLR, Java CC, Xtext, entre otros.

Palabras clave: DSL, ANTLR, Xtext, Java CC, lenguaje declarativo, gramática, modelo semántico.

1. INTRODUCCION

En los últimos años, los lenguajes de dominio específico DSL (Domain Specific Language) han tomado gran importancia en el desarrollo de productos de software, ofreciendo una gramática reducida y aplicable únicamente al dominio específico para el cual el DSL ha sido diseñado. Adicionalmente, la evolución que se ha generado alrededor de la construcción de herramientas para el diseño e implementación de DSLs ha permitido que cada día sea mucho más simple crearlos.

Así mismo, una dirección interesante es construir DSLs cuyo dominio específico sea un ambiente de ejecución con un conjunto de características y reglas bien definidas. De esta manera, se permite la libertad de diseñar cualquier lenguaje de dominio específico en cualquier herramienta y lenguaje. Cada uno de estos lenguajes desarrollados puede ser aplicado a diferentes elementos mediante el desarrollo de código fuente que es soportado por cada DSL, en el ambiente de ejecución establecido como dominio

2. LENGUAJES DE DOMINIO ESPECIFICO

El propósito de un Lenguaje de dominio específico consiste en tener un lenguaje de programación que pueda resolver situaciones de un dominio en particular [1], de tal forma que no pueden resolver problemas presentes en otros dominios. Hay dos clases de DSLs que son internos y externos.

Un DSL interno se basa de un lenguaje existente. Su objetivo es proveer mayor capacidad o potencialidad a un lenguaje mediante librerías que contienen el DSL interno. Esta clase de DSLs también son llamados “Fluent interface” o “Fluent API”

Un DSL externo es un lenguaje independiente que cuenta con una sintaxis y un interpretador propio. En muchos casos los DSLs externos proveen un IDE propio el cual puede ser conformado por un editor grafico y/o un editor de texto.

Los DSLs ofrecen ventajas como las siguientes:

- Expresan soluciones con base en los términos relacionados con el dominio.
- La solución obtiene el nivel de abstracción apropiado para el dominio.
- Los expertos de dominio fácilmente deben poder comprender y desarrollar programas basados en un DSL.
- Permiten validaciones a nivel del dominio.

Los DSLs pueden presentar desventajas como las siguientes:

- La curva de aprendizaje sobre un nuevo lenguaje puede ser desfavorable en relación con la aplicabilidad limitada al dominio

- El costo de diseñar, implementar y mantener un DSL puede ser elevado, además el numero de herramientas para desarrollar DSLs es reducido
- Al crear un DSL se hace necesario crear una herramienta que permita el desarrollo de programas para dicho DSL. Esta herramienta puede llegar a ser un editor de texto o un editor grafico que incrementa el costo de desarrollo de los componentes para poner en producción el DSL.

Los DSLs pueden ser construidos mediante el paradigma de programación declarativa o el paradigma de programación imperativa.

Un DSL declarativo, está diseñado para que el código fuente de un programa basado en este DSL sea escrito mediante la especificación de declaración de proposiciones o afirmaciones que describen el problema y detallan su solución. Los resultados obtenidos como consecuencia de la ejecución de una declaración se realizan mediante mecanismos internos de control incluidos en el intérprete del DSL.

Un DSL imperativo realiza la ejecución de un programa basado en los estados del mismo y en instrucciones que cambian el valor de estos estados. Entonces un programa imperativo es un conjunto de instrucciones ejecutables secuencialmente que permiten realizar una tarea determinada.

Alrededor de Java, existen varios lenguajes y herramientas para la construcción de DSLs como ANTLR, Xtext y Java CC.

ANTLR (ANother Tool for Language Recognition) es una herramienta de lenguaje que provee un framework para la construcción de reconocimiento, interpretación, compilación y translación de descripciones gramaticales que contienen acciones basadas en un lenguaje objetivo como C, C++, C#, Java, entre otros [2]. ANTLR provee un buen soporte para arboles de sintaxis abstracta, descubrimiento de errores y reporte de errores. Además permite la generación de parsers y lexers.

ANTLR genera analizadores pred-LL(k), y él mismo utiliza un analizador pred-LL(k) para leer los archivos en los que están escritas las reglas EBNF. ANTLR admite acciones en sus reglas, además de otras prestaciones como paso de parámetros, devolución de valores o herencia de gramáticas [4].

Xtext es un ambiente de desarrollo basado en Eclipse que permite la creación de DSLs con capacidad de edición mediante IDEs de Java. Xtext provee APIs para describir los aspectos del DSL mediante la notación EBNF. Basado en la gramática escrita para un lenguaje en particular, Xtext genera una implementación completa en Java [5]. La interpretación en Xtext se facilita mediante el uso de diferentes patrones que capturan el código fuente de un programa basado en el DSL creado, capturando línea por línea ejecutando las instrucciones escritas.

Xtext provee un edito de texto robusto y descubrimiento de errores tanto en la gramática como en el código fuente de los programas que se basan de un DSL.

Java CC es un generador de parser y generador de analizador léxico para usar con aplicaciones Java. El generador de parser es una herramienta que lee una gramática y la convierte en un programa de Java que puede reconocer diferentes puntos de la gramática [6].

3. CASO DE ESTUDIO

Para el caso de estudio se ha planteado elaborar un proyecto que permita programar tanques de guerra mediante diferentes DSL. Para ello, se requiere una aplicación en java que se comporta como infraestructura que provee diferentes servicios que puedan soportar la funcionalidad descrita en los DSLs.

El proyecto se divide en dos componentes principales que son infraestructura y DSL.

La infraestructura consiste en un API de servicios extensible, en la cual se deben considerar los siguientes componentes:

- Arena: es el escenario en donde se visualizan los tanques y demás elementos involucrados en el juego como bloques y balas. Los atributos que posee la arena son ancho, alto. Los servicios que posee la arena son generar reporte de comportamiento del juego, cargar escenarios pre-configurados, cargar información de equipos.
- Equipo: está conformado por tanques y DSLs. Posee un nombre y un color el cual se le aplica a todos los tanques pertenecientes al equipo.
- Tanque: hace referencia a un tanque de guerra el cual puede desplazarse hacia adelante y puede rotar sobre su propio eje en sentido reloj y contra-reloj. El tanque posee un cañón el cual podrá rotar de forma independiente al tanque. A través del cañón, el tanque podrá disparar balas a diferentes objetivos que se encuentren ubicados en la arena. Los atributos que posee el tanque son posición en x, y, orientación, cañón, numero de balas, nivel de energía. Los servicios que posee el tanque son desplazar hacia adelante, rotar sentido reloj, rotar sentido contra-reloj, rotar el cañón sentido reloj, rotar el cañón sentido contra-reloj, disparar, recargar balas, modificar nivel de energía, detectar objetivo.
- Bala: es el elemento por el cual se ataca al equipo contrario. Un impacto de bala sobre un tanque reduce el nivel de energía. Cada tanque tiene un número de balas el cual se agota y se recarga con un pequeño tiempo de retardo.
- Boque: es un elemento inactivo que se comporta como obstáculo en la arena. La infraestructura es la encargada de controlar las colisiones que tienen los tanques y balas con los bloques.
- JarClassLoader: permite cargar en memoria dinámica los componentes jar requeridos por cada equipo para la ejecución de sus DSLs.
- DSL: permite la ejecución del DSL asignado a cada tanque y cargado a través del JarClassLoader. Este DSL requiere el uso de un descriptor el cual contiene el código fuente requerido para la interpretación del DSL Dicho código fuente es cargado y ejecutado con base en los jar requeridos.

La figura 1 presenta el diagrama de clases de la infraestructura el cual permite evidenciar la estructura de este proyecto y las relaciones entre sus elementos previamente descritos.

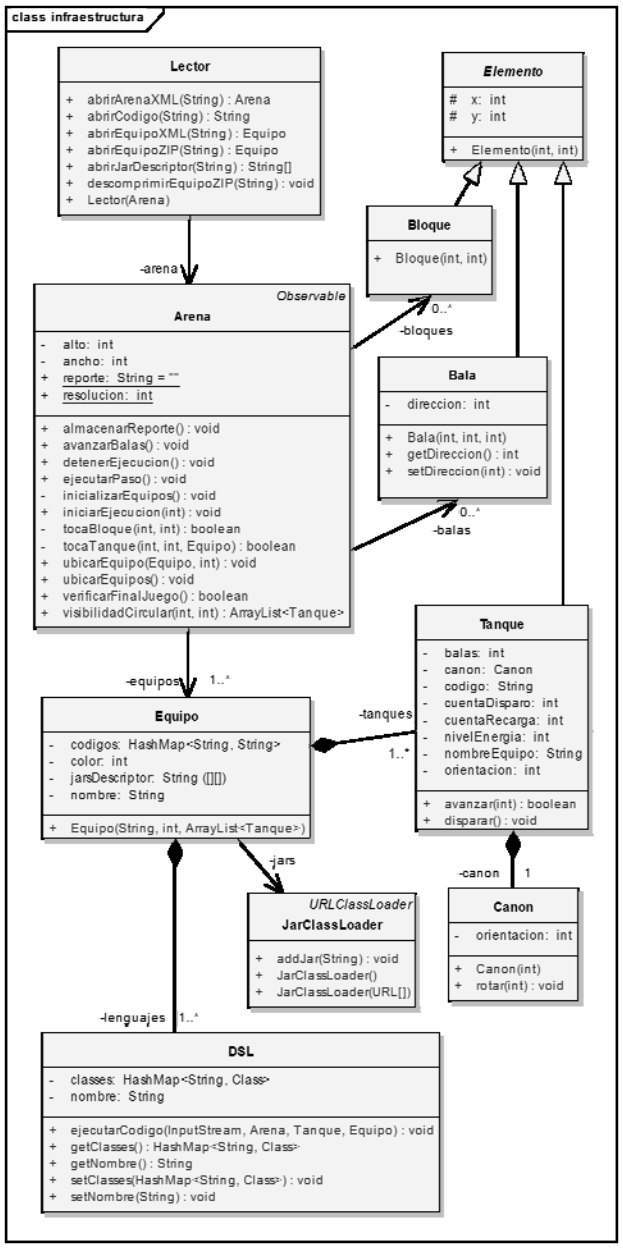

Fig 1. Diagrama de clases de infraestructura

La configuración de la arena se realiza mediante un archivo XML. Este archivo contiene los siguientes atributos:

- Ancho y el alto de la arena en pixeles
- Numero de balas que se asigna por defecto a cada tanque. En el momento en que un tanque agota este numero de balas, el numero de balas de dicho tanque se reinicia con este valor pasado un tiempo de retardo.
- Nivel de energía de cada uno de los tanques presentes en la arena. Cuando un tanque recibe un impacto de bala, este valor decrece en 1. Cuando el nivel de energía de un tanque es 0, este tanque queda eliminado y desaparece de la arena.
- Resolución de los elementos de la arena. Este valor indica el tamaño de cada bloque y de cada tanque de la arena.
- Bloque. Se puede configurar cualquier número de bloques. Cada bloque debe contener las coordenadas "x" y "y" dentro de la arena.

El código XML para la configuración de la arena es el siguiente:

```
<arena ancho="640" alto="480" balas="10" nivelEnergia="5"
resolucion="20">
  <bloque x="500" y="100"/>
  <bloque x="350" y="150"/>
  <bloque x="330" y="210"/>
  <bloque x="250" y="180"/>
  <bloque x="370" y="280"/>
  <bloque x="180" y="210"/>
</arena>
```

Al cargar la arena en la infraestructura, se coloca la resolución y los bloques configurados. La figura 2 visualiza el resultado de la configuración de la arena.

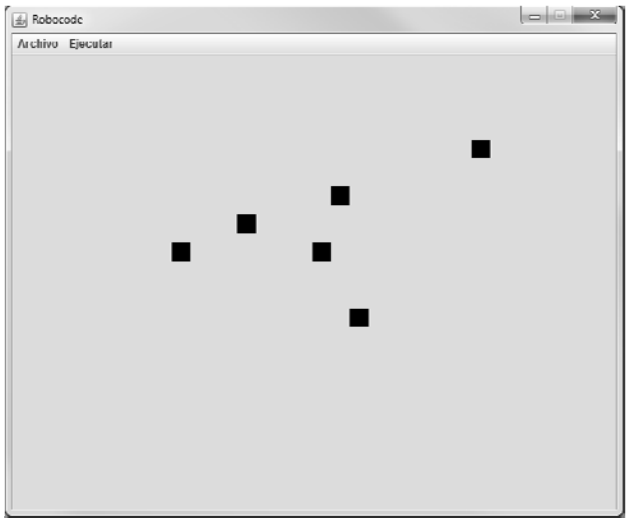

Fig 2. Resultado de la configuración de la arena

Posteriormente se realiza la carga de cada uno de los equipos. Cada equipo contiene un conjunto de archivos los cuales están reunidos en un archivo comprimido. El número de archivos incluidos en el archivo comprimido puede variar de acuerdo a la cantidad de DSLs definidos y sus respectivos archivos de soporte para la interpretación de dichos DSLs.

En cualquier caso como mínimo se requiere los siguientes archivos:

- XML del equipo. Contiene los siguientes elementos
  - Nombre: se comporta como el identificador del equipo
  - Color: es requerido para dibujar los tanques del equipo en la arena. El color es un numero entero equivalente al formato RGB
  - Tanques: un equipo puede contener hasta cuatro tanques. Cada tanque tiene los siguientes atributos:
    - Orientación: define la orientación en grados del tanque
    - Lenguaje: define el archivo jar del DSL con el cual dicho tanque va a ser ejecutado.
    - Código: define el código fuente soportado por el DSL definido en el atributo anterior para la ejecución del tanque.
    - Cañón: establece la orientación del cañón del tanque.

  El código XML para la configuración de un equipo con cuatro tanques es el siguiente:

```
<equipo nombre="EquipoAzul" color="5592575">
  <tanque orientacion="120"
lenguaje="RoboCodeDSLBasico.jar" código="miCodigo1.dsl">
    <canon orientacion="0" />
  </tanque>
  <tanque orientacion="140"
lenguaje="RoboCodeDSLBasico.jar" código="miCodigo1.dsl">
    <canon orientacion="40" />
  </tanque>
  <tanque orientacion="160"
lenguaje="RoboCodeDSLBasico.jar" código="miCodigo2.dsl">
    <canon orientacion="80" />
  </tanque>
  <tanque orientacion="180"
lenguaje="RoboCodeDSLBasico.jar" código="miCodigo2.dsl">
    <canon orientacion="120" />
  </tanque>
</equipo>
```

- Descriptor. Contiene información acerca de los archivos que debe cargar la infraestructura al cargar el equipo. Las siguientes líneas de texto son un ejemplo para cargar un equipo en donde su archivo se denomina “equipo.xml”, un DSL en donde su archivo se denomina “RoboCodeDSLBasico.jar”, dos códigos fuente en donde sus archivos se denominan “miCodigo1.dsl” y “miCodigo2.dsl” y finalmente un archivo que soporta la interpretación de los códigos anteriores en el DSL anterior.

```
equipo>>equipo.xml
dsl>>RoboCodeDSLBasico.jar
código>>miCodigo1.dsl
código>>miCodigo2.dsl
jar>>antlr-3.3.jar
```

- Librerías. Son archivos jar que contienen el soporte para interpretar los DSLs
- Descriptor de librería. Contiene las líneas de código necesarias para interpretar el DSL.
- DSL. Debe ser un archivo jar con el cual se pueden ejecutar los códigos fuentes de cada tanque.
- Código fuente. Es el código ejecutable mediante el DSL cargado.

Para el caso de estudio en cuestión, un equipo se compone de un archivo comprimido que contiene los archivos de la siguiente figura

Fig 3. Elementos de un equipo en archivo comprimido

Una vez cargados los equipos que pueden ser máximo seis, el juego de batalla está listo para iniciar. El objetivo del juego de batalla es que un equipo logre quedar con al menos un tanque. Este equipo será el ganador.

En cada tanque se visualiza dos números que corresponden al nivel de energía y al número de balas. Al cargar cuatro equipos el juego se visualiza como se muestra en la figura 4.

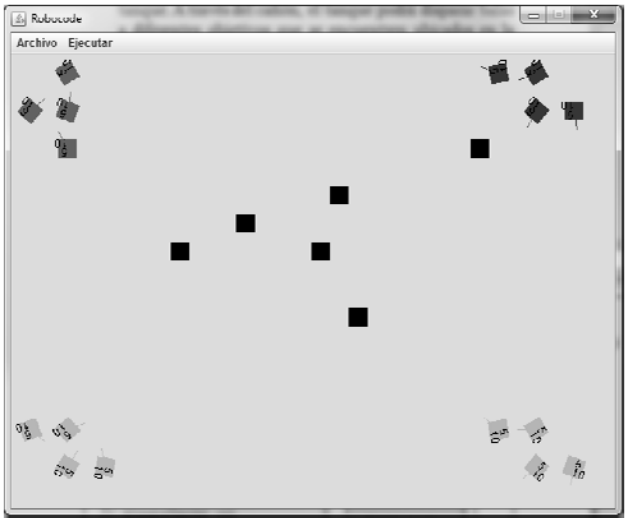

Fig 4. Resultado de la carga de cuatro equipos

Al ejecutar el juego, los diferentes tanques empiezan a comportarse de acuerdo al código fuente definido para cada uno de los tanques. Este código fuente es interpretado por el correspondiente DSL el cual es soportado por las librerías requeridas y especificadas en el descriptor que se ubica en el archivo comprimido del equipo. La figura 5 muestra un ejemplo de un instante de la ejecución del proyecto con los cuatro equipos anteriormente cargados.

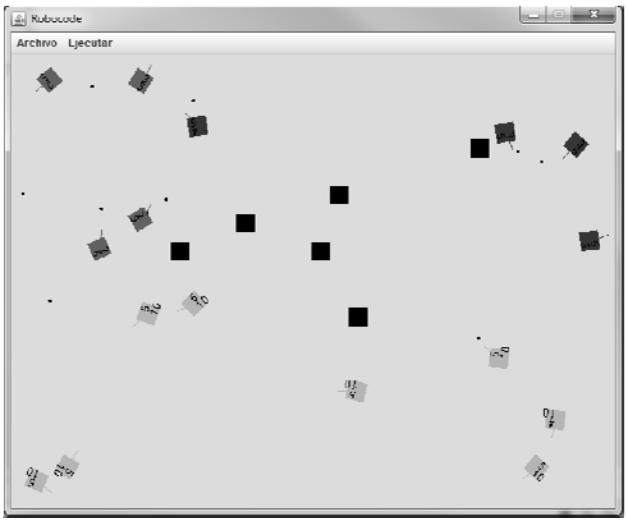

Fig 5. Ejecución del juego

Al finalizar la ejecución del juego, se presenta un reporte el cual indica el progreso del juego y el equipo ganador. En este reporte se indica el momento en que ocurre un evento. Los eventos pueden ser “Tanque de equipo destruido” indicando que el nivel de energía de un tanque de un equipo es 0 y “Equipo derrotado” indicando que el nivel de energía de todos los tanques es 0. De esta manera la ejecución de la aplicación se detiene en el momento en que todos los equipos menos uno hayan sido destruidos.

La figura 6 muestra un reporte de ejecución de la aplicación con cuatro equipos en donde cada equipo posee cuatro tanques.

```
Jan 11 17:42:06 2012>> Inicio de ejecucion
Jan 11 17:42:10 2012>> Tanque de equipo EquipoRojo destruido
Jan 11 17:42:11 2012>> Tanque de equipo EquipoAzul destruido
Jan 11 17:42:18 2012>> Tanque de equipo EquipoRojo destruido
Jan 11 17:42:18 2012>> Tanque de equipo EquipoAzul destruido
Jan 11 17:42:19 2012>> Tanque de equipo EquipoRojo destruido
Jan 11 17:42:19 2012>> Tanque de equipo EquipoAzul destruido
Jan 11 17:42:21 2012>> Tanque de equipo EquipoAmarillo destruido
Jan 11 17:42:21 2012>> Tanque de equipo EquipoVerde destruido
Jan 11 17:42:24 2012>> Tanque de equipo EquipoRojo destruido
Jan 11 17:42:24 2012>> Equipo EquipoRojo derrotado
Jan 11 17:42:30 2012>> Tanque de equipo EquipoVerde destruido
Jan 11 17:42:30 2012>> Tanque de equipo EquipoVerde destruido
Jan 11 17:42:31 2012>> Tanque de equipo EquipoAmarillo destruido
Jan 11 17:42:41 2012>> Tanque de equipo EquipoVerde destruido
Jan 11 17:42:41 2012>> Equipo EquipoVerde derrotado
Jan 11 17:42:45 2012>> Tanque de equipo EquipoAzul destruido
Jan 11 17:42:45 2012>> Equipo EquipoAzul derrotado
Jan 11 17:42:45 2012>> Fin del juego
Jan 11 17:42:45 2012>> Detencion de ejecucion
```

Fig 6. Reporte de ejecución

4. CONCLUSIONES

Los DSLs se han convertido en herramientas poderosas que permiten el desarrollo de aplicaciones para un dominio particular mediante una gramática simple y autodocumentada usable para cualquier tipo de usuario que conozca el dominio para el cual es creado un DSL.

La construcción de un proyecto que actúe como dominio, permite la creación e interpretación de diferentes DSLs que se basen de dicho dominio permitiendo la interacción entre los componentes del dominio que se desenvuelven mediante programas escritos en los diferentes DSLs

El caso de estudio se basa en un proyecto de infraestructura el cual es extensible mediante el uso de reflexión en java. Esta característica permite la carga en memoria dinámica de los DSLs y de los diferentes componentes requeridos para la ejecución de dichos DSLs.

Para lograr la interacción de los diferentes DSLs, es necesario otorgar de manera secuencial el uso del procesamiento a cada uno de los elementos dentro de la ejecución del proyecto. Esta característica se ha logrado mediante el uso de temporizadores en java que se basa del procesamiento de hilos.

5. REFERENCIAS

[1] Fowler Martin. Domain Specific Languages. 2011

[2] Parr Terrence. The definitive ANTLR Reference, Building Domain Specific Languages. 2007

[3] Parr Terrence. Language implementation patterns, Create Your Own Domain-Specific and General Programming Languages. 2010.

[4] García Enrique, Troyano José. Guía práctica de ANTLR. 2003

[5] XTEXT. http://www.eclipse.org/Xtext/documentation/

[6] JavaCC. http://javacc.java.net/

[7] Amyot Daniel, Farah Hanna, Roy Jean-Francois. Evaluation of Development Tools for Domain-Specific Modeling Languages.

[8] Santos André, Koskimies Kai, Lopes Antónia. Automating the construction of domain-specific modeling languages for object-oriented frameworks. The Journal of Systems and Software. 2010.