Evaluación analítica de una herramienta de visualización de traductores dirigidos por la sintaxis

Ángel F. Sánchez-Granados & Jaime Urquiza-Fuentes
Departamento de Informática y Estadística
Universidad Rey Juan Carlos
Móstoles, España
{angelfrancisco.sanchez,jaime.urquiza}@urjc.es

Adrián García-Oller & José M. Loeches-Ruiz
Departamento de Informática y Estadística
Universidad Rey Juan Carlos
Móstoles, España
{a.garciaoll,jm.loeches}@alumnos.urjc.es

***Resumen*—Este trabajo presenta una herramienta educativa basada en tecnologías de visualización. El principal objetivo de esta herramienta es servir de apoyo a los estudiantes en el aprendizaje de los conceptos asociados a la traducción dirigida por la sintaxis. En primer lugar, se ha desarrollado una API para permitir la anotación de la especificación del traductor. En segundo lugar, la ejecución del traductor genera como efecto colateral la traza de ejecución que será interpretada por la interfaz de visualización. Así, se ofrece una representación visual de la ejecución del traductor en términos de las acciones semánticas ejecutadas y los valores de los atributos de los símbolos. Debido a la separación entre la API de anotación y la interfaz de visualización, se provee cierto grado de independencia entre la interfaz y la herramienta de generación del traductor. Con la actual situación por la COVID-19, no se ha podido realizar una evaluación empírica. En su lugar, se ha realizado una evaluación analítica desde trespuntos de vista: la usabilidad de la interfaz, la usabilidad de la API y la efectividad pedagógica. Los resultados son positivos y nos llevan hacia la evaluación empírica de la herramienta definiendo futuras líneas de trabajo.**

***Index Terms*—Visualización del software, Procesadores de lenguajes, Traductores dirigidos por la sintaxis, Educación en CS**

I. Introducción

La disciplina de la visualización del software aplicada al entorno educativo tiene un largo recorrido, sirva como ejemplo las animaciones de algoritmos de ordenación realizadas por Ronald Baecker en los años 60 [1]. Una de las posibles aplicaciones que podemos encontrar de esta especialidad es la visualización de conceptos abstractos, que nos permitirá conseguir que los estudiantes sean capaces de formar un mejor modelo mental del concepto a estudiar. Por otro lado, tanto Naps et al. [2] como Hundhausen et al. [3] concluyen que además de la visualización *per se* es determinante la forma en que esta se usa por parte de los estudiantes.

En la enseñanza de la Informática, la asignatura de Procesadores de Lenguajes resulta habitualmente compleja para los estudiantes debido en parte a la dificultad de aplicar a situaciones reales los conceptos abstractos que se trabajan [4]. Uno de los temas fundamentales que la componen es la traducción dirigida por la sintaxis (TDS), que se basa en conceptos de analizadores sintácticos enriquecidos con la posibilidad de producir información , transmitirla a través del árbol sintáctico enriquecido y ejecutar un lenguaje de programación [5], y que resulta de gran importancia a la par que complejo. La TDS proporciona, por tanto, la base teórica para entender algunas etapas de los compiladores, como pueden ser la generación de código intermedio o la comprobación de tipos. Precisamente, la visualización pretende servir de enlace entre los conceptos teóricos y su aplicación práctica.

Este trabajo se centrará en la aplicación de la visualización del software a la TDS con el objetivo de que los estudiantes mejoren su comprensión. El resto del artículo se estructura como sigue. La sección II describe los trabajos relacionados sobre la visualización de Procesadores de Lenguajes. La herramienta se describe en la sección III. La sección IV describe la evaluación preliminar de la herramienta. Finalmente, la sección V expone las conclusiones y los trabajos futuros.

II. Trabajos relacionados

El desarrollo de traductores dirigidos por la sintaxis (en adelante traductores) es una tarea compleja y por ello existen una gran variedad de herramientas diseñadas para apoyarla. Algunas se centran exclusivamente en el desarrollo, por ejemplo CUP[1] o ANTLR[2], aunque también se usan en entornos educativos. El esquema de funcionamiento básico consiste en que el desarrollador produce una especificación del traductor, las herramientas antes mencionadas se usan para generar el código fuente del traductor a partir de la especificación. Una vez generado dicho traductor, este se ejecutará como cualquier otra aplicación procesando la cadena de entrada y realizando las tareas incluidas en la especificación.

Por otra parte, también existe software relacionado donde la visualización asume un rol determinante, como los sistemas LISA, CUPV, VCOCO y EvDebugger.

LISA [6] es una herramienta de generación de intérpretes y compiladores que dispone de una especificación gramatical propia. La versión 2.2 soporta analizadores de tipo LL, LR y LALR. Su planteamiento se centra en el desarrollo de lenguajes de dominio específico, por lo que su enfoque no

Este trabajo ha sido financiado por los proyectos del Ministerio de Economía y Competitividad (ref. TIN2015-66731-C2-1-R) y la Comunidad de Madrid e-Madrid-CM (P2018/TCS-4307). El proyecto e-Madrid-CM se financia con fondos estructurales (FSE and FEDER).

[1]http://www2.cs.tum.edu/projects/cup/, 2021
[2]https://www.antlr.org/, 2021

es educativo. La especificación del traductor se debe realizar con una notación propia de la herramienta. Su interfaz gráfica permite visualizar el árbol sintáctico, el grafo de dependencias y los valores de los atributos del árbol enriquecido. A pesar de tener un enfoque profesional y no haber sido evaluada formalmente, existen experiencias donde los estudiantes reflejan una opinión positiva de la herramienta [6] además de potenciar su participación activa en las clases [7].

CUPV [8] es una herramienta de visualización para las gramáticas CUP, de modo que aprovecha su especificación para la visualización, y está orientada específicamente para el entorno educativo. Permite visualizar atributos del proceso de compilación tales como valores semánticos, reducciones, el árbol sintáctico o la entrada. Actualmente no se encuentra disponible para su descarga.

VCOCO [9] es un software que permite visualizar la ejecución de traductores dirigidos por la sintaxis generados con la herramienta COCO/R[3]. Por ello, utiliza su especificación léxica, sintáctica y de TDS. Esta herramienta permite visualizar la información léxica y sintáctica relativa a la instrucción de código que está en ejecución. Su interfaz de visualización usa el enfoque de un depurador. Esta herramienta tampoco se encuentra disponible para su descarga.

EvDebugger [10] es una herramienta de corte educativo que permite diseñar traductores con la ayuda de un depurador visual. Con una especificación gramatical propia, a través de su depurador se puede visualizar el árbol sintáctico con sus nodos de atributo, los atributos de la gramática y los valores semánticos de los atributos calculados. Su evaluación recogió buenas opiniones de estudiantes y profesores pero no se ha podido encontrar enlace de descarga para esta herramienta.

Es obvio que ha habido un interés por utilizar las tecnologías de visualización con los traductores. De las herramientas que hemos encontrado, las que más se acercan a nuestro propósito son LISA y EvDebugger, aunque existen ciertas lagunas por cubrir. Ambas utilizan un lenguaje de especificación propietario, la primera no está diseñada para el ámbito educativo aunque se haya hecho una evaluación referente al analizador sintáctico y la segunda no se puede descargar para su utilización.

Nuestro objetivo se centra en conseguir una herramienta que pueda visualizar los detalles de la traducción dirigida por la sintaxis y que sea tan independiente como sea posible de las herramientas de desarrollo de traductores, de forma que no haya que aprender un nuevo lenguaje de especificación ni ceñirse a un único tipo de generador de traductores.

III. Descripción de la herramienta

Como ya se ha mencionado, el principal objetivo de esta herramienta es visualizar el funcionamiento de un traductor dirigido por la sintaxis. Aun así, también se quiere conseguir que estas visualizaciones sean independientes de las herramientas de desarrollo de traductores.

[3]http://www.ssw.uni-linz.ac.at/Coco/, 2021

En la Fig. 1 se pue ver un esquema del uso y la arquitectura de este sistema. Nuestro sistema se compone de una interfaz de visualización y una API de generación de información a visualizar. La utilización sería de la siguiente forma. En primer lugar, el usuario escribe su especificación del traductor en cualquiera de los lenguajes de generación soportados por la herramienta (1), y a continuación se anota con llamadas al API de generación dando lugar a la *especificación anotada* (2). Se genera el traductor utilizando la especificación anotada (3), cuando este se ejecuta con una cadena de entrada también proporcionada por el usuario (4), produce como efecto lateral una traza de ejecución en formato XML (5) que es la información que la interfaz utilizará para mostrar la visualización de la ejecución del traductor (6).

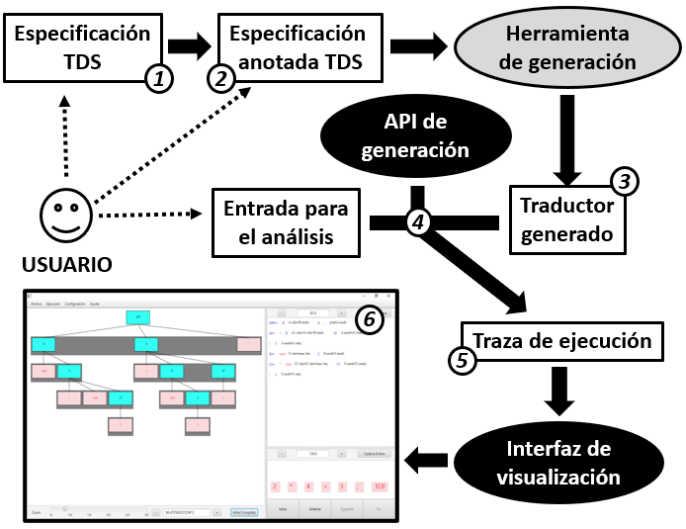

Figura 1. Arquitectura del sistema.

Por lo tanto, el sistema se basa en la visualización post-mortem, mostrando la ejecución del traductor una vez esta haya terminado. A continuación, se describe el diseño tanto de la interfaz como de la API de generación de la traza de ejecución.

III-A. Interfaz de visualización

La ejecución de un traductor dirigido por la sintaxis depende del modelo de evaluación de las acciones semánticas asociadas a las producciones de la gramática. Nuestro sistema utiliza la evaluación en tiempo de compilación [11], de forma que se necesita tener en cuenta el tipo de analizador que se está utilizando. Por ello, será necesario que la visualización muestre el proceso de construcción del árbol sintáctico enriquecido con los atributos de los símbolos gramaticales y las acciones semánticas ejecutadas que manipulan y modifican el valor de dichos atributos. Por otro lado, la ejecución de un traductor necesita de una cadena de entrada.

Todo ello nos lleva a identificar tres elementos principales en nuestra interfaz de visualización: la especificación del traductor, la entrada con que se ejecutará el traductor y el estado del árbol sintáctico enriquecido en cada momento. La

especificación del traductor es necesaria puesto que es lo que el usuario trata de comprender, el elemento que él debe aprender a generar. La entrada es quien proporcionará la información sobre el momento de la ejecución en que se encuentra el traductor. Y el árbol sintáctico enriquecido es propiamente la visualización del estado interno del traductor, que se asemeja al modelo mental utilizado generalmente en la enseñanza de procesadores de lenguajes [11]. Siendo estos tres elementos los fundamentales para la compresión de la ejecución de un traductor, su visualización debe realizarse de forma simultánea y sincronizada.

El diseño de la interfaz ha seguido un proceso de prototipado seguido de su evaluación utilizando técnicas de evaluación heurística [12]. Durante todo el proceso se contó con la participación de estudiantes familiarizados con la asignatura. Después de reflexionar sobre los diferentes elementos de la interfaz, se celebró una sesión de prototipado participativo contando con el profesor de Procesadores de Lenguajes y cuatro estudiantes con conocimientos contrastados de la asignatura. El resultado fue un conjunto de esquemas en los que se plasman ideas sobre la sincronización y manipulación del árbol sintáctico enriquecido y la especificación: al seleccionar una producción en el árbol se debería destacar en la especificación y viceversa, la posibilidad de ocultar/mostrar las acciones semánticas de la especificación según el usuario las necesite, la sincronización de la cadena de entrada con el árbol enriquecido, así como la sincronización mostrando los valores de los atributos tanto en el árbol enriquecido como en las acciones semánticas.

La siguiente fase de prototipado se encargó de producir el esquema de las tres áreas de trabajo en la interfaz. Están asociadas con los tres elementos principales de la visualización: árbol sintáctico enriquecido, especificación del traductor y la cadena de entrada. Además, se han diseñado las diferentes visualizaciones para símbolos terminales y no terminales, los valores de los atributos y los controles de navegación de la ejecución. La implementación de esos prototipos se desarrolló utilizando Java Swing para la interfaz genérica y JGraph [4]para la manipulación y visualización del árbol sintáctico enriquecido. Los detalles sobre este primer proceso están disponibles en [13].

Una vez se tuvo implementada la primera versión de la interfaz se realizó una evaluación heurística [12] que contó con la participación de seis estudiantes diferentes de los que participaron en el prototipado, pero también con conocimientos contrastados de la asignatura. Las principales mejoras que se realizaron gracias a estas evaluaciones se centraron en aspectos como la interacción básica con los botones de navegación, el formato de presentación de las acciones semánticas dentro de la especificación, las capacidades de configuración de la visualización así como el mantenimiento visible del punto de interés del usuario (último elemento actualizado en el árbol). Estas mejoras coincidieron con la reimplementación de la interfaz usando JavaFX [5], que además de ofrecer algunas ventajas en el desarrollo de la interfaz permite una mejor generación de los nodos del árbol. Los detalles sobre esta segunda fase de diseño se puede encontrar en [14].

Resumiendo las características de la interfaz, el sistema permite la visualización dinámica de la ejecución de un traductor dirigido por la sintaxis centrándose en los aspectos fundamentales: cadena de entrada a procesar, proceso construcción del árbol sintáctico (debido a la evaluación en tiempo de compilación), especificación del traductor (reglas sintácticas y acciones semánticas), y finalmente el árbol enriquecido con los atributos. El estado mostrado en las tres vistas de la visualización –cadena de entrada, árbol sintáctico enriquecido y especificación del traductor– está sincronizado. Estos aspectos se pueden ver en la fig. 2. Además se proporcionan otras características más básicas pero muy útiles como la capacidad de configuración de la interfaz asegurando coherencia en la representación de símbolos terminales y no terminales en todas las vistas, zoom sobre el árbol enriquecido o la navegación mediante teclado, ratón o incluso selección de un elemento concreto de la cadena de entrada.

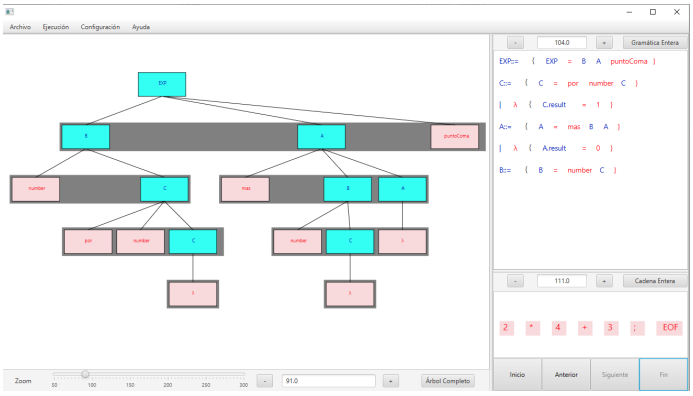

Figura 2. Captura de pantalla de la interfaz de visualización

III-B. API de generación

El objetivo de la API es generar la información de la ejecución del traductor, la traza de ejecución, que será interpretada y visualizada por la interfaz antes descrita. La traza de ejecución se generará en formato XML, según la estructura básica definida en [13]. Se divide en 3 partes: en la primera se identifica la especificación indicando las reglas gramaticales (el elemento `<translator>`), los símbolos y las acciones semánticas. La segunda representa al árbol sintáctico (el elemento `<stree>`) mediante nodos relacionados entre sí. La tercera parte (el elemento `<content>`) completa la información de cada nodo del árbol sintáctico (con referencias a los elementos XML del árbol dentro de `<stree>`) con las acciones semánticas ejecutadas, los valores de los atributos involucrados y el estado de procesamiento de la cadena de entrada. La figura 3 muestra un fragmento de una traza de la ejecución del traductor.

[4]JGraph - https://github.com/jgraph, 2021

[5]JavaFX - https://openjfx.io/, 2021

```
... <translator>
      <type>bottom-up</type>
      <rule id="R1">
        <symbol> ...</symbol>
        ...
      </rule>
      ...
    </translator>
    <stream>2 * 4 + 3 ; </stream>
    <stree>
      <num_nodes>17</num_nodes>
      <height>5</height>
      <node id="0">
        <element>number</element>
        <level>3</level> <terminal>true</terminal>
      </node>
      ...
    </stree>
    <content>
      <step id="0">
        <type>desplazamiento</type>
        <newrule refRegla="R6">
          Regla B = number C }
        </newrule>
        <stream>
          <read>2 </read>
          <pending>* 4 + 3 ; </pending>
        </stream>
        <element>number</element>
        <value>number.vlex=2</value>
      </step>
      ...
    </content> ...
```

Figura 3. Ejemplo de contenido en una traza de ejecución en el fichero XML

La generación de esta traza se produce como efecto lateral de la ejecución del traductor anotado con llamadas a la API generadora. El proceso de anotación consiste principalmente en añadir llamadas a los métodos de la API dentro de las acciones semánticas de la especificación del traductor. El diseño de la API persigue principalmente dos objetivos, por un lado la facilidad de uso y por otro la cobertura de los dos tipos de analizadores: ascendentes y descendentes. Para ello, hemos trabajado con dos herramientas ampliamente utilizadas: CUP para los analizadores ascendentes, y ANTLR para los descendentes. Aunque los métodos utilizados en ambos tipos de analizadores son similares, existen pequeñas diferencias debido por ejemplo a que CUP solo permite atributos sintetizados mientras que ANTLR permite utilizar tanto atributos sintetizados como heredados.

El diseño de la API se realizó en tres fases, una descripción más detallada se puede encontrar en [13], [14]. En primer lugar se analizaron todas las llamadas necesarias para generar la traza XML. Así, para una regla de la especificación hay que añadir la información al XML de: la especificación (reglas, símbolos, acciones semánticas y atributos) y la aplicación particular de la regla durante la construcción del árbol enriquecido, tanto a nivel sintáctico como a nivel de traducción con la ejecución de las acciones semánticas y los valores concretos de cada atributo involucrado. La segunda fase se centró en minimizar el número de llamadas que el usuario debería incluir para anotar la especificación. Este proceso consideró dos aspectos, aunar llamadas a métodos y automatizar procesos totalmente mecánicos como el de creación de clases para los símbolos de la especificación (que ayudan a manipular la información de la traza correspondiente cada símbolo) y su inclusión en la especificación mediante el comando:

```
java -jar VisTDSApiCreator.jar <especificacion.cup>
```

Finalmente, durante la tercera fase se revisaron todas las llamadas para asegurar que la información que el usuario debe proporcionar en cada llamada a los métodos de la API sea la mínima imprescindible. Al incluir en la especificación estas anotaciones de forma manual, los docentes podrían tanto mostrar valores de atributos como proporcionar explicaciones de útiles que mejoren la comprensión del funcionamiento del traductor. A continuación se describen las anotaciones para los analizadores ascendentes (con CUP) y descendentes (con ANTLR).

III-B1. Anotación de especificaciones ascendentes con CUP: La ejecución del comando `VisTDSApiCreator` también añade a la definición de los símbolos terminales su especificación como String que permite su visualización en la interfaz. Después, basta con añadir las siguientes llamadas en las producciones de la gramática. Si la producción no corresponde a una regla borradora se añade la siguiente llamada:

```
writer.addStepNonTerminal( LHS-name,
    synthesized-attr-name, attr-value, text);
```

Si la regla es borradora, el método usado es `writer.addStepLambda` con los mismos argumentos. Un ejemplo del uso de estos métodos se puede ver en la fig. 4.

```
A ::= plus B:b A:a1 {: RESULT=b + a1; :}
|{: RESULT=0; :};
```

(a) Fragmento de especificación CUP sin anotar

```
A ::= plus B:b A:a1 {:
    writer.addStepNonTerminal("A", "result", b + a1,
        "A::= + B A1 {A.result=B.result+A1.result;}");
    RESULT=b + a1;
:}

|{:
    writer.addStepLambda("A", "result", 0, "{A.result=0;}");
    RESULT=0;
:};
```

(b) Fragmento de especificación CUP anotada

Figura 4. Ejemplo de especificación CUP: (a) sin anotar y (b) anotada

III-B2. Anotación de especificaciones descendentes con ANTLR: Después de ejecutar el comando `VisTDSApiCreator` hay que usar las siguientes llamadas para anotar la gramática. Al ser un analizador descendente, el árbol se crea utilizando un recorrido en profundidad y necesitamos realizar dos anotaciones una al comienzo de la regla y otra al final. La primera se realiza con la siguiente llamada:

```
writer.addStepNonTerminal(LHS-name,
    inherited-attr-name, synthesized-attr-name,
    inherited-attr-reference);
```

La segunda llamada tiene el siguiente formato:

```
writer.updateNonTerminals(rule,
```

```
    synthesized-attr-reference,
    first-sybol-reference);
```

Al igual en el caso de CUP, las reglas lambda también tienen una anotación especial:

```
writer.addStepLambda(LHS-name, inherited-attr-name,
    synthesized-attr-name, inherited-attr-reference,
    rule);
```

En la Fig. 4 se muestra un ejemplo de anotación de este tipo de especificaciones.

```
exp returns [Exp exp0]:
    b0=b
    a0=a[Integer.parseInt(((ExpContext)_localctx).b0.b0.getValue())]
    ';'
    {
        _localctx.exp0=exp0;
    }
 ;
```

(a) Fragmento de especificación ANTLR sin anotar

```
exp returns [Exp exp0]:
   {
       Node node=writer.addStepNonTerminal("EXP", null, null, "null");
   }
   b0=b
   a0=a[Integer.parseInt(((ExpContext)_localctx).b0.b0.getValue())]
   ';'
   {
       writer.updateNonTerminals(
           "EXP::= B {A.valor=B.result;} A ; {print(A.result);}",
           ((ExpContext)_localctx).a0.a0.getValue(),
           ((ExpContext)_localctx).b0.b0);
       _localctx.exp0=exp0;
       writer.writeXML();
   }
;
```

(b) Fragmento de especificación ANTLR anotada

Figura 5. Ejemplo de especificación ANTLR: (a) sin anotar y (b) anotada

IV. Evaluación preliminar

La evaluación de la herramienta estaba planificada para el segundo semestre del curso 2019-2020. Debido a las restricciones impuestas por la pandemia COVID-19, no se pudo plantear un experimento controlado que permitiera realizar la mencionada evaluación, siendo postpuesto al segundo semestre del curso 2020-2021.

Aún así, se ha contado con las impresiones recogidas de estudiantes con conocimientos contrastados sobre la asignatura. Esta recolección de información se ha realizado durante el proceso de diseño y desarrollo de la interfaz. Los estudiantes se mostraron entusiasmados con la posibilidad de ver una representación de la ejecución del TDS. También hicieron multitud de comentarios que han servido para mejorar diversos detalles de la interfaz. Es importante destacar que, al ser un tema muy complejo de la asignatura, el nivel de exigencia para con la herramienta por parte de los estudiantes es muy alto.

Por ello, hemos decidido realizar en su lugar una evaluación analítica. Además de las evaluaciones heuríscas de la interfaz mencionadas anteriormente, contemplamos la evaluación analítica desde otros dos puntos de vista: la utilización de la API por parte de los docentes y los contenidos ofrecidos por la herramienta a los estudiantes.

IV-A. Evaluación de la API, punto de vista docente

De entre los distintos enfoques existentes para evaluar la usabilidad de una API hemos escogido el propuesto en [15] puesto que se adapta mejor a las restricciones anteriormente mencionadas. En nuestro caso, los profesores serán los usuarios de la API y la evaluación aplicará un conjunto de reglas heurísticas contemplando los siguientes aspectos: facilidad de aprendizaje, productividad del programador, prevención de errores, simplicidad de uso, consistencia y adecuación al modelo mental del programador. A continuación, se describen las características de la API según las heurísticas de [15].

La segunda regla *Match between system and the real world* pone el foco en los nombres de clases y métodos de la API. En este caso se cumple dado que los nombres de los métodos se asocian directamente a las acciones que realizan: `addStepLambda(...)` para la producción lambda y `addStepNonTerminal(...)` para el resto de producciones. La cuarta regla *Consistency and standards* se cumple al dar los mismos nombres a los métodos que realizan las mismas tareas para los diferentes tipos de analizadores (descendentes y ascendentes), así como al usar los mismos atributos y en el mismo orden para métodos asociados al mismo tipo de analizador. La sexta regla *Recognition rather than recall* se cumple al asignar nombres cortos a los métodos, siendo estos además claros y entendibles cuando el IDE los muestra en los popups de autocompletado. La octava regla *Aesthetic and minimalist design* también se cumpliría al carecer de métodos redundantes y tener restringidos los argumentos a los indispensables. Finalmente, la décima regla *Help and Documentation* se cumple al ofrecer un manual de uso de la API.

IV-B. Evaluación pedagógica, punto de vista de los estudiantes

Suponiendo cumplidos los requisitos de usabilidad, existen estudios que tratan sobre la eficacia educativa de herramientas que usan representaciones visuales. Uno de los más importantes describe los principios para el diseño de materiales educativos multimedia [16]. Estos principios describen requisitos que deberían cumplir las herramientas educativas para que los estudiantes consigan un aprendizaje adecuado. A continuación se analiza la herramienta según estos principios.

El principio *Multimedia* se cumple dado que la ejecución del TDS muestra la visualización del árbol junto con las explicaciones incluidas mediante API. El principio *Spatial contiguity* se cumple dado que la visualización del árbol semántico muestra explicaciones junto al correspondiente elemento, producción o símbolo. El principio *Coherence* está estrechamente relacionado con la heurística *Aesthetic and minimalist design* dado que sólo se muestran los elementos estrictamente necesarios a los estudiantes. El principio *Personalization* se cumple con las explicaciones incorporables a las anotaciones, siendo preferible un estilo conversacional frente a un estilo formal. El principio *Interactivity* se cumple dado que los usuarios pueden controlar el paso y la dirección de la visualización, además de poder mostrar las explicaciones por

pantalla. Finalmente, el principio *Signaling* se cumple puesto que los nuevos elementos en la visualización son claramente visibles cuando aparecen. Además, cuando el usuario posiciona el cursor sobre una producción en el árbol semántico, la correspondiente producción y acciones semánticas se resaltan en la especificación del TDS.

Finalmente, explicamos los principios que no se pueden utilizar al evaluar esta aplicación: *Temporal contiguity* ya que el sistema sólo muestra explicaciones cuando el usuario lo solicita; *Modality* y *Redundancy* dado que no se han contemplado narraciones en el sistema. Tampoco se ha contemplado el desarrollo de este software como un sistema adaptativo según la interacción y el estilo de aprendizaje de los estudiantes.

V. Conclusiones y trabajo futuro

En este trabajo se describe una herramienta que permite visualizar la ejecución de traductores dirigidos por la sintaxis partiendo de sus especificaciones, creadas por los propios usuarios (tanto profesores como alumnos) y con sus propias cadenas de entrada. Resultados preliminares en campos relacionados [3], [7], [10] nos indican que el uso de la visualización de software en la enseñanza de la traducción dirigida por la sintaxis puede ayudar a su aprendizaje.

Tras analizar las herramientas relacionadas existentes se ha podido comprobar que ninguna es capaz de trabajar con diferentes notaciones de especificaciones, algunas se restringen a un determinado tipo de analizador sintáctico [8], [9] mientras que otras tienen limitaciones de interacción y/o despliegue [6], [10], barreras que pretenden eliminarse con esta propuesta.

Con esta herramienta los usuarios pueden anotar de forma transparente sus propias especificaciones recibiendo feedback sobre su funcionamiento, sin necesidad de disponer de una batería de ejemplos previos. A nivel de desarrollo, se ha concebido para ser flexible tanto para la adaptación de nuevas notaciones de especificaciones como para ser independiente de la interfaz de visualización, teniendo la capacidad de generar la especificación de salida adaptada a las necesidades del visualizador utilizado. Gracias a la API, los textos añadidos a la aplicación de producciones y valores de atributos permiten a los profesores variar el nivel de las explicaciones proporcionadas desde un mínimo nivel a modo de depurador hasta un nivel de explicación textual detallada. Estas animaciones se pueden utilizar tanto en clase como en el tiempo de estudio propio de los estudiantes, algo muy necesario en el contexto educativo actual de restricciones impuestas por la COVID-19. Como el progreso de la animación está totalmente controlada por el usuario, el profesor podría hacer preguntas sobre lo que acontecería en algún momento concreto, los valores de determinados atributos o las acciones semánticas a que se ejecutarían. Por otro lado, los estudiantes pueden reproducir tantas veces como deseen la animación, así como centrarse únicamente en los puntos que les interesa de la ejecución del traductor dirigido por la sintaxis.

La versión actual de la herramienta carece de una evaluación formal empírica. Sin embargo, durante su diseño se ha contado con las impresiones de 10 estudiantes con conocimientos contrastados de la asignatura. Además de ayudar a detectar mejoras, mostraron una opinión muy positiva sobre dos aspectos: la posibilidad de *ver* el comportamiento del traductor y la interactividad ofrecida por la herramienta.

La línea de trabajo futura más importante se centra en la anotación automática de las especificaciones de traductores. Sin embargo, antes realizaremos una evaluación empírica de la interfaz. Por otro lado, la API debe permitir la adaptación a cualquier herramienta de generación de TDSs. Finalmente, se afrontará la liberación del código fuente para que esté a disposición de la comunidad de desarrolladores, así como el desarrollo de una versión cloud de esta herramienta que pueda ser utilizada en cualquier lugar y desde cualquier dispositivo sin necesidad de instalar ningún tipo de software.

Referencias

[1] R. Baecker, "Interactive computer-mediated animation," Ph.D. dissertation, M.I.T. Department of Electrical Engineering, April 1969.

[2] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen ..., and J. A. Velázquez-Iturbide, "Exploring the role of visualization and engagement in computer science education," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 131–152.

[3] C. Hundhausen, S. Douglas, and J. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages and Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[4] M. Hewner, "Undergraduate conceptions of the field of computer science," in *Proc. Ninth ACM Conference on International Computing Education Research*. New York, NY, USA: ACM, 2013, pp. 107–114.

[5] D. E. Knuth, "Semantics of context-free languages," *Mathematical systems theory*, vol. 2, pp. 127–145, 06 1968.

[6] M. Mernik and V. Zumer, "An educational tool for teaching compiler construction," *IEEE Transactions on Education*, vol. 46, no. 1, pp. 61–68, 2003.

[7] D. Nicolalde-Rodríguez and J. Urquiza-Fuentes, "Educational impact of syntax directed translation visualization, a preliminary study," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018, pp. 313–314.

[8] A. Kaplan and D. Shoup, "Cupv—a visualization tool for generated parsers," in *Proc. Thirty-First SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 11–15.

[9] R. D. Resler and D. M. Deaver, "Vcoco: A visualisation tool for teaching compilers," in *Proc. 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education: Changing the Delivery of Computer Science Education*, ser. ITiCSE '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 199–202.

[10] D. Rodríguez-Cerezo, P. R. Henriques, and J. Sierra, "Attribute grammars made easier: Evdebugger a visual debugger for attribute grammars," in *2014 International Symposium on Computers in Education (SIIE)*, 2014, pp. 23–28.

[11] A. Aho, M. Lam, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[12] J. Nielsen and R. Molich, "Heuristic evaluation of user interfaces," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 249–256.

[13] J. M. Loeches, "Herramienta de visualización de la traducción dirigida por la sintaxis," B.S. Thesis, Escuela Técnica Superior de Ingeniería Informática. Universidad Rey Juan Carlos, 2018.

[14] A. García, "Visualización de la traducción dirigida por sintaxis con vistds y vistds api," B.S. Thesis, Escuela Técnica Superior de Ingeniería Informática. Universidad Rey Juan Carlos, 2019.

[15] B. Myers and J. Stylos, "Improving api usability," *Commun. ACM*, vol. 59, no. 6, pp. 62–69, May 2016.

[16] R. Mayer, *Multimedia Learning*. Cambridge University Press, 2008.