VolText

User manual

Colombi Simone

Gambarara Alberto

Scarpellini Stefano

Indice

1. Introduction

VolText is a project developed in the context of the course of Formal Languages and Compilers. The goal is the design and implementation of a Java library for generating a PDF document and a graphical interface that allows simplified use of it. The most inherent part of the course was the creation of a grammar that describes the structure of a PDF document using the typical constructs of HTML syntax, such as div, images, texts and lists, but using a syntax similar to JSON.

The library is particularly focused on developing documents that have simple graphics such as flyers, small announcements, notices, business cards, etc.

The tool doesn't want and doesn't act as a replacement for LaTeX but takes the essential elements, creating a different version both in terms of language and in terms of components, eliminating formulas, graphics, tables, etc. not useful for the development of graphic documents.

2. Grammar

2.1. General description

The grammar has been designed and implemented thinking about who works with web languages and are familiar with HTML/JSON.

Using the typical syntax of the JSON language for the construction of custom tags, the grammar has been structured as a PDF document composed of various elements that can be drawn inside in positions and sizes at will.

The grammar is reported and, for each rule, its function is described.

```
grammar VolText;

@header {
    package antlr;
}

pdf :               A pdfattr* (stylesheet)? (page)+ C;
pdfattr:            'title:' STRING ENDNLINE
        |           'author:' STRING ENDNLINE
        |           'path:' STRING ENDNLINE;
stylesheet:         'stylesheet' O element* C;
element:            '@' STRING O attrStyle* C;
attrStyle:          'cross-point:' NVAL ENDNLINE
        |           'shape:"' ('RECTANGLE' | 'CIRCLE' | 'TRIANGLE') ENDLINE
        |           ('fit-x' | 'fit-y') ':' TFVAL ENDNLINE
        |           ('pos-x' | 'pos-y') ':' NOTVAL? NVAL (UNIT)? ENDNLINE
        |           'angle-rotation' ':' NOTVAL? NVAL ENDNLINE
        |           ('height' | 'width') ':' NVAL (UNIT)? ENDNLINE
        |           ('p_height' | 'p_width') ':' NVAL ENDNLINE
        |           'ordered:' TFVAL ENDNLINE
        |           'bullet:' STRING ENDNLINE
        |           ('font-family:' | 'font-family-ttf:' | 'font-family-otf:') STRING
                    ENDNLINE
        |           'font-size:' NVAL ENDNLINE
        |           ('bold:' |   'italics:' |'underline:') TFVAL ENDNLINE
        |           'colorT-bullet:' STRING ENDNLINE
        |           'color-bullet:' COLORVAL ENDNLINE
        |           'colorT:' STRING ENDNLINE
        |           'color:' COLORVAL ENDNLINE
        |           'position:' POSVAL ENDNLINE
        |           'alignment:' ALIGNVAL ENDLINE
        |           'orientation:' ORIENTATION ENDNLINE
        |           'oob:' TFVAL ENDNLINE
        |           'format:' FORMATVAL ENDNLINE;
page:               'page' O pageattr* pae* C;
pae:                elemd | div;
div:                'div' O (color | idval | imganumber | elemd | positionv | fitAttr |
                    figure | tvalue)* C;
tvalue:             'cross-point:' NVAL ENDNLINE;
figure:             'shape:"' ('RECTANGLE' | 'CIRCLE' | 'TRIANGLE') ENDLINE;
```

```
elemd:              text
    |               list
    |               img;
img:                'img' O imgattr* imgElem imgattr* C;
imgattr:            (idval | imganumber | positionv | fitAttr);
imgElem:            'URL:' STRING ENDNLINE;
list:               'list' O (listattr | listElem)* C;
listElem:           'item:' STRING ENDNLINE;
text:               'text' O txtattr* txtElem* txtattr* txtElem* txtattr* C;
txtattr:            (color | idval | imganumber | positionv | alignment | txtval |
                    fitAttr);
txtElem:            'string:' STRING ENDNLINE;

//ATTRIBUTES

fitAttr:    (        'fit-x'
            |        'fit-y') ':' TFVAL ENDNLINE;
imganumber: (         'pos-x'
            |         'pos-y' ) ':' NOTVAL? NVAL (UNIT)? ENDNLINE
            |         'angle-rotation' ':' NOTVAL? NVAL ENDNLINE
            |         ('height'
            |         'width') ':' NVAL (UNIT)? ENDNLINE;
idval:               'id:' STRING ENDNLINE;
listattr:           'ordered:' TFVAL ENDNLINE
        |           'bullet:' STRING ENDNLINE
        |           idval
        |           imganumber
        |           txtval
        |           positionv
        |           color
        |           colorBullet
        |           fitAttr;
txtval:             ('font-family:'
        |            'font-family-ttf:'
        |            'font-family-otf:') STRING ENDNLINE
        |           'font-size:' NVAL ENDNLINE
        |           ('bold:'
        |           'italics:'
        |           'underline:') TFVAL ENDNLINE;
colorBullet:        'colorT-bullet:' STRING ENDNLINE
        |           'color-bullet:' COLORVAL ENDNLINE;
color:              'colorT:' STRING ENDNLINE
        |           'color:' COLORVAL ENDNLINE;
positionv:          'position:' POSVAL ENDNLINE;
alignment:          'alignment:' ALIGNVAL ENDLINE;
pageattr:           'orientation:' ORIENTATION ENDNLINE
        |           'oob:' TFVAL ENDNLINE
        |           'p_width' ':' NVAL ENDNLINE
        |           'p_height' ':' NVAL ENDNLINE
        |           'format:' FORMATVAL ENDNLINE
        |           idval;

//TERMINALI

NOTVAL:             '-';
UNIT:               'mm' | '%' | 'pt';
FORMATVAL:          'A0'
        |           'A1'
        |           'A2'
        |           'A3'
```

```
          |              'A4'
          |              'A5'
          |              'A6';
ORIENTATION:             ('hor' | 'ver');
COLORVAL:                '#' ([0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]
                         [0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]);
ALIGNVAL:                 'left'
          |               'center'
          |               'right'
          |               'justify';
TFVAL:                   'true'
          |              'false';
POSVAL:                  'lt'
          |              'ct'
          |              'rt'
          |              'lc'
          |              'cc'
          |              'rc'
          |              'lb'
          |              'cb'
          |              'rb' ;
NVAL:                    [0-9]+('.'[0-9]+)?;
ENDLINE:                 '";';
ENDNLINE:                ';';
O :                      ':{';
A :                      '{';
C :                      '}';
STRING:                  '"' (~[\t\r\n])+ '"';
WS :                     [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

The first rule, *pdf*, allows the opening of the document, the insertion of attributes such as title, author and saving path of the pdf, relating to the document, the possibility to insert a part of the document style, one or more pages and the closing of the document.

A summary structure of the *stylesheet* tag was made as

```
stylesheet:{
        @IdElement1:{
                    << attributes for that element >>
        }
}
```

The page can contain various attributes such as orientation, oob (out-of-bound), width, height and format, as specified in the following table:

attributo	valori possibili
orientation	hor, ver
oob	true,false
p_width	decimal number
p_height	decimal number
format	A0, A1, A2, A3, A4, A5, A6

Each page contains inside divs, therefore blocks that may also contain other elements, or elements themselves such as images, lists and texts. These elements and their attributes will be treated on *chapter 2.2*.

A page, as also reported in the ISO 32000-1 specification, section 11.4.7, must have a white background color. So, if you want to completely color a page, you can insert a div with dimensions fitted respect to the page or insert other elements (in this version, given the choice of not having internal containers into others, it is not possible to create different ones in a page with color of background different than white, precisely because it is already a colored container).

2.2. Page elements

Now we treat the various elements that can compose a page. We will treat the images, texts and lists that can be inserted referring to the size of the page. Finally, we will discuss the div, more complex elements and which may contain within them the other types of item, but with measuring references related to the div in which they are encapsulated. Each attribute of each element must always be closed by a ; (semicolon) while the closure of the elements is realized by a right brace.

2.2.1. Images

An image must be inserted through the "*img: {<<attributes>>}*" tag.

It may contain attributes placed in position even after the only mandatory attribute, namely the URL. The *url* is a string that specifies the relative or absolute path of that image inside the computer. If relative, the reference will start from the folder where the text document is located and, as we will see later, if a document is not opened but only the text is pasted into the GUI, we will have errors regarding the lack of the object at the specified address.

The various attributes available for images are now listed:

- *id*, the identifier of the element inside the document. This has been designed for use especially in the style sheet.
- *pos-x* and *pos-y*, indicate the position of the element on the horizontal and vertical axis. The value can be positive or negative and also contain the unit of measurement (millimeters (mm), percentage (%) or points (pt), by default mm).
- *angle-rotation*, indicates the rotation of the element respect to its center in degrees. If the specified value is positive, a counterclockwise rotation will be performed, on the contrary with a negative value a clockwise rotation will be performed.

- *height* and *width*, indicate the height and width of the image in case you need to resize the element contained on the page or div. They too can be expressed in millimeters, percentages or points.
- *position*, contains a pair of letters that allow you to insert the element in its container (page or div) in a predetermined position (imagining a subdivision into 9 squares of the container). They can therefore be *l*, *c* and *r* to indicate whether the element must be respectively on the left, centered or right in the container, and *t*, *c* or *b* to indicate whether the element must be at the top (*t*), central (*c*) or bottom (*b*).
- *fit-x* and *fit-y*, contains a boolean value (true, false) to indicate if the element must be resized over the entire size of the container (be it the page or the div) respect to the x or y axis.

2.2.2. Texts

A text can be entered via the tag "*text: {<<attributes>>}*".

It can contain attributes mixed with text strings without precedence constraints.

The strings/chapters are contained in "*string*" attributes and are written in the PDF in the order that is used in writing of the *text* element.

The attributes of the text element are the following:
- *id,* as already described for the images, identifies the object.
- *colorT* and *color*, attributes that report the color of the text (unique for all strings). *color* specifies a color in hexadecimal #RGBA format, where R is the red color, G the green color, B the blue color and A the alpha, which is the transparency value of the element. In the texts and in the lists, the alpha is not considered, contrary to what happens for the div. *colorT* specifies a color in the form of text (black, blue, red, green, …). The last attribute that is declared will be used in the drafting of the document.
- *pos-x, pos-y, angle-rotation, height, width, position, fit-x* and *fit-y*: see the discussion in chapter 2.2.1 on images.
- *alignment*, used for aligning the text (*left*, *center*, *right* and *justify*).
- *font-family, font-family-ttf* and *font-family-otf*, , specify the font to be used in the drafting of the document. If you are using *font-family* you can use standard fonts such as helvetica, courier and times. If you are using *font-family-ttf*, then you can specify the path where the file with extension ttf (True Type Font) that describes the font to use is available. If the folder where the font resides also contains the relative font in bold, italic and / or bold italic then the path to specify will be that of the regular font (ending with Regular.ttf, and in which

the other files will end with Bold.ttf, Italic.ttf and BoldItalic.ttf, respectively for bold, italic and bold italic): in this case any texts or parts of text in bold (written as *\bold text in bold \bold*), italic (written as *\italic text in italic \italic*) or bold italic (written as *\bold \italic text in bold italic \italic \bold*). The same rules work to the font-family-otf attribute, to which fonts with the extension otf (Open Type Font) apply.

- *font-size*, contains the font size to use in the texts.
- *bold, italic* and *underline*, are boolean attributes that allow you to specify whether the entire text should be written in bold, italic or underlined (the latter specification has been implemented but not completed due to difficulties in integrating the PDFBox-layout library with PDFBox).

2.2.3. List

A list can be entered through the tag *list: {<<attributi>>}*
It can contain attributes mixed with text strings (list items, *item*) without precedence constraints.
The attributes of a list can be:

- *id*, like the previous elements.
- *ordered*, this is used to indicate whether it is an ordered list of items (numeric list) or whether it is an unsorted list (points list).
- *bullet*, specifies the type of "point" that starts the list item. The possible points are *odd* (full dot, •), *even* (double arrow, »), or *new(Bullet)* (in that case *Bullet* is a user-decided string that can be used as a point).
- all text-related attributes except alignment are also implemented for lists.
- *colorT-bullet* and *color-bullet*, attributes that indicate the bullet color (unique to all bullets). *color-bullet* specifies a *#RGBA* color where R is red, G is green, B the blue and A the alpha value (the transparency value of the element). The alpha value is not considered in this case. *colorT-bullet* denotes a color as text (black, blue, red, green, …). the last attribute that is declared will be used in the drafting of the document.

## 2.2.4. Div

div can be inserted through the tag *div: {<<attributi e/o elementi>>}*.

The div, as also described in html, is a container that allows you to group elements within it. In this version of the project, a div can contain only images, texts, and lists, but not other containers. The container can be customized through various attributes, and the elements that are inside it take the size of the container as a dimensional reference.

may contain attributes mixed with elements (*img*, *text* e *list*) without precedence constraints.

Now we analyze the various attributes of a container:

- *id,* like any element, specifies an identifier that allows to recognize the element.
- *color* e *colorT*, as previously presented regarding texts and lists, specifies the filling color of the container. A breakdown in the color of the fill and color of the border has not been realized, which can be inserted in future developments. *color* denotes the color in *#RGBA* format and in this case the alpha (A) specifies the transparency of the container, to provide the possibility to see other elements underlying it. *colorT* instead allows the use of colors by color names.
- *pos-x*, *pos-y*, *angle-rotation*, *height*, *width*, *fit-x* and *fit-y*, as mentioned in the previous sections, specify the position, angle of rotation, size and fit with respect to the horizontal and vertical axis.
- *position*, the position of the container relative to the page on which it is inserted without the need to specify spatial coordinates.
- *shape*, the shape of the container, between *RECTANGLE*, *CIRCLE* and *TRIANGLE*. In the case of the rectangle the position and size values will be used to draw it. In the case of the circle, position and size values will be used to draw a circle or ellipse. in the case of the triangle, in addition to position and size, you will need to insert a cross-point attribute expressed as a percentage that specifies the H point specified in the figure:
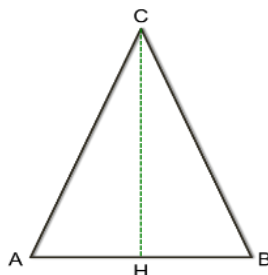
2.3. Translator implementation

The translator has been implemented through the usage of Listener. While we were aware of the advantages of Visitor Design Pattern, we opted for the usage of Listener because of methods that it implemented. Whether the visitor has generic access to the nodes, that is, you cannot infer whether they are inside the incoming node or are inside the outgoing node, the Listener, instead, allows this splitting and therefore more specific access to the node.

The following is a part of the analysis tree taken from the grammar in Chapter 4.3 for the text element:

```
text:{
        id:"TESTO2";
        colorT:"black";
        position:ct;
        height:30mm;
        string:"Prova";
        angle-rotation: 45;
}
```
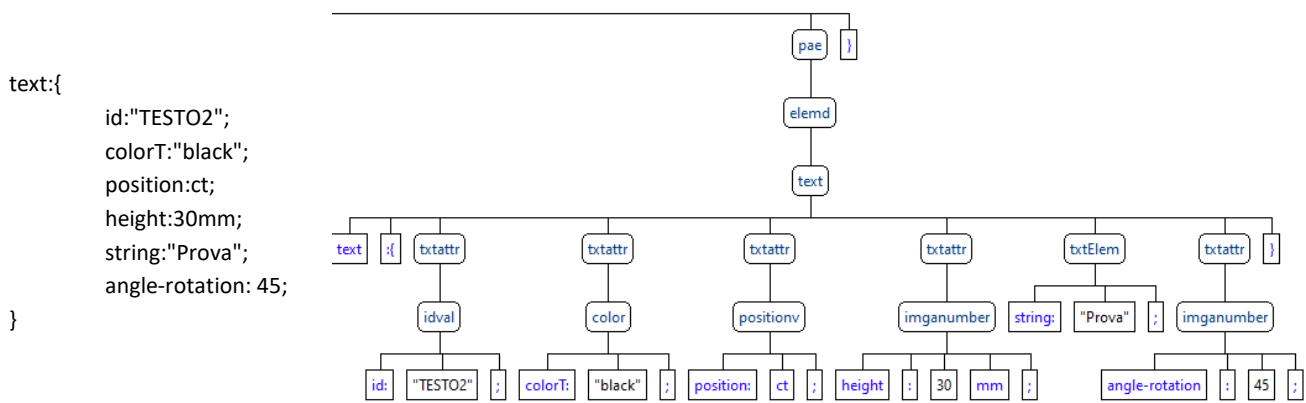

2.4. Grammatical choices

The main choice is due to the grammar processed to implement the style sheet. This is a choice that is convenient for implementing attributes directly as children of *attrStyle* as children of the elements on the pages.

The way grammar reads the sheet, the position of the *stylesheet* as the second part of the nonterminal *pdf* does not allow you to have an idea of the element to which the specifications refer, since the identifier in the style sheet refers to a generic object, and therefore does not verify the correctness of the individual attributes. because you do not have any control over the stylesheet you can use an id for multiple elements of a different nature, and you can insert attributes even if they do not belong to the type of object that you want to customize.

2.5. Grammatical errors

Let us now deal with the various errors and situations that may arise in the drafting of a text document and which the programme detects and interprets. Under not much more circumstances does the program report an error but only show warnings to report a typo or lexical error.

The first case of error (which will also be treated as an example of an error in *Chapter 4*) is the case of an oversight of a period and comma (or a brace). In this case, the user is told that a symbol is missing in a specific line/location, but the system still manages to continue redaction. This is the typical case of a lexical error.

The same applies even if an attribute does not belong to an element (such as a *bullet* attribute for the *img* tag). This is the typical case of syntactic error.

A report that the program returns is also referred to the wrong path where the image is located by identifying with the *url* of the image. In fact, the system specifies that it failed to read the image in that specific path. However, the system avoids loading the image and continues with the continuation of the analysis of the text.

The same "error" is reported if the specified path is not for a file for using specific fonts. The latter two examples refer to semantic errors.

Other reports are made even if the inserted position of the item is incorrect or outside the page, if the page is specified both as a format and size (in that case the page format is taken), or if the size is larger than the container that contains the item.

3. Used tools

Several tools have been used for the development of the project:

- *GitHub*: GitHub is a Version Control service used in our project to track changes implemented in our software. This also helped the almost real-time sharing of code for better group work.
- *GitHub Desktop*: client that allows a simplified and rapid use of GitHub, also useful to track changes and versions of the project.
- *ANTLR 4*: ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. From a grammar, ANTLR generates a parser that can build and analyze analysis trees. Version 4 has been used to keep up with the latest versions and for convenience with the IDE Eclipse, given the presence of a special plugin.
- *Eclipse*: IDE for library and GUI development, which also supports various plugins for ANTLR and JavaFX.
- *PDFBox*: java library for creating and editing PDF documents.
- *PDFBox-layout*: additional java library to PDF-box that allows simplified (not entirely compatible) management of texts and lists.
- *JavaFX*: library that permits the implementation of a graphical interface for a java program. For Eclipse, the *e(fx)clipse* plugin was used that allows the creation of JavaFX projects and the use of attached libraries.
- *Launch4j*: software for encapsulation of highly customizable exe executables from jar executables.

4. User interface

The application can be used through a graphical interface developed with JavaFX. At the execution it comes with a GUI consisting of a text area where the user can write and edit the grammar of a PDF document, two buttons similar to each other, respectively for opening and saving a text file, a red button for creating the document, and a non-editable text area that is used to communicate any messages and errors to the user.
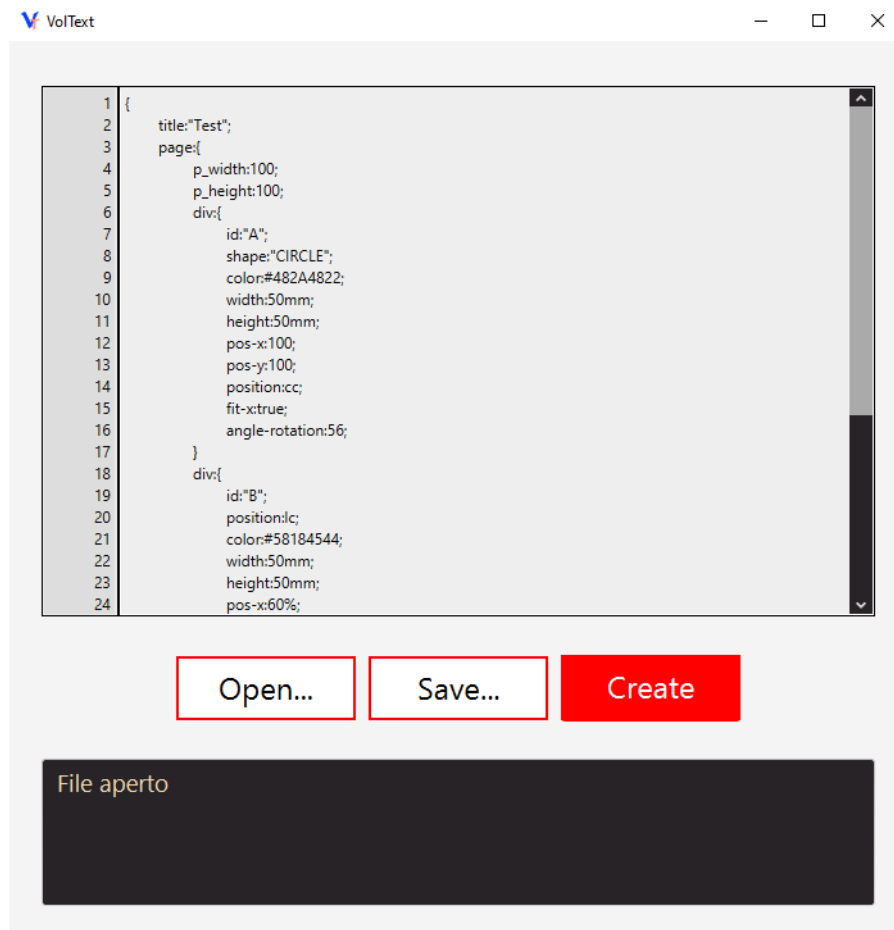
4.1. Setup

In order to launch the program, it is just needed to download an updated version of the Java Development Kit from 12 onwards (if not detected is required when starting the executable VolText.exe) and copy into the bin folder of the downloaded JDK (typically C:\Program Files\ Java\jdk-xxx\bin) the files located in the lib_to_java_bin folder (they are the same as in the VolTextGUI> javafx-sdk-11.0.2> bin folder). It is recommended to keep a copy of the files already present in the bin folder of JDK in order to restore them in case of deletion of the program.

4.2. General use case

The user can write the grammar from scratch, but the most common use case is certainly the one in which the grammar, or a part of it, is already in an existing text file. In this case, with the "Open ..." button is possible to select the file to be loaded through a file system navigation interface, receiving a confirmation message.
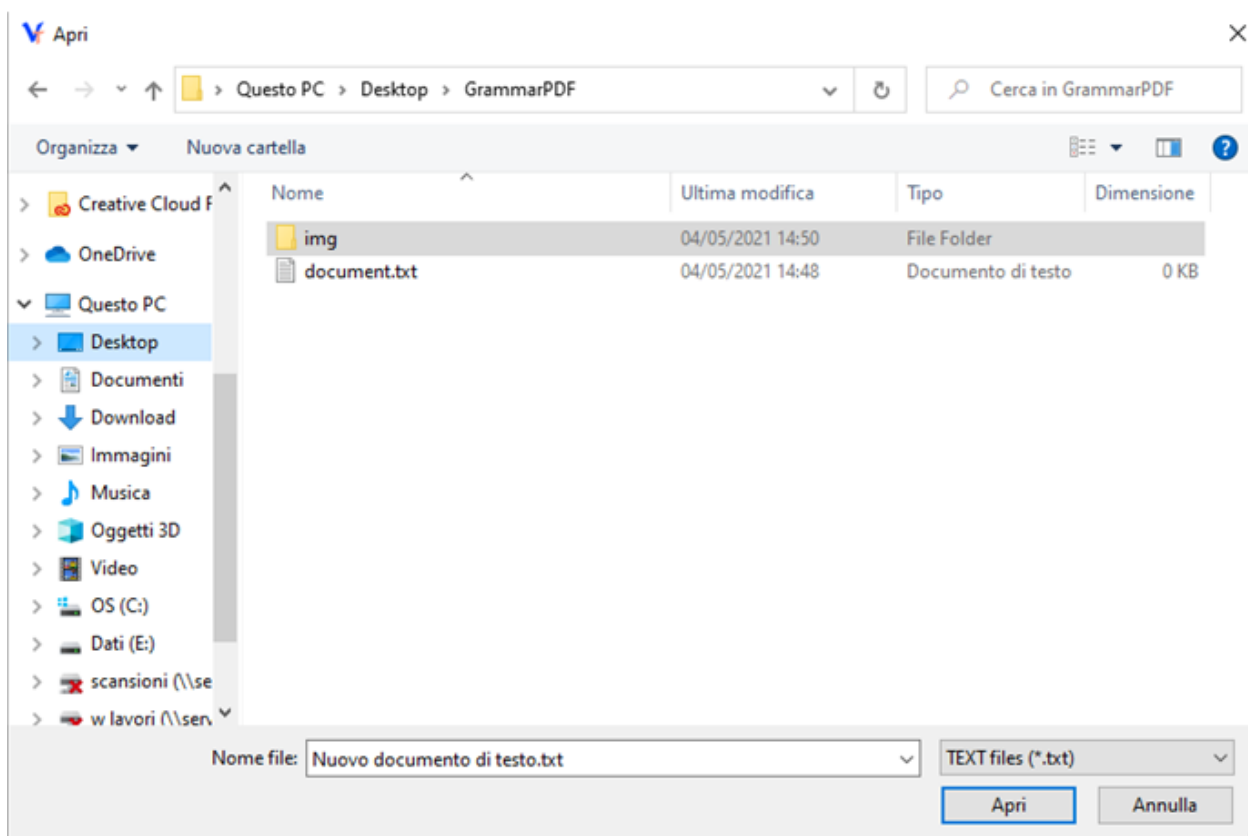
After opening the file, the user can make all the necessary changes to the grammar, and by pressing the "Save ..." button it can be saved to a file using an interface similar to the one used during the loading process.
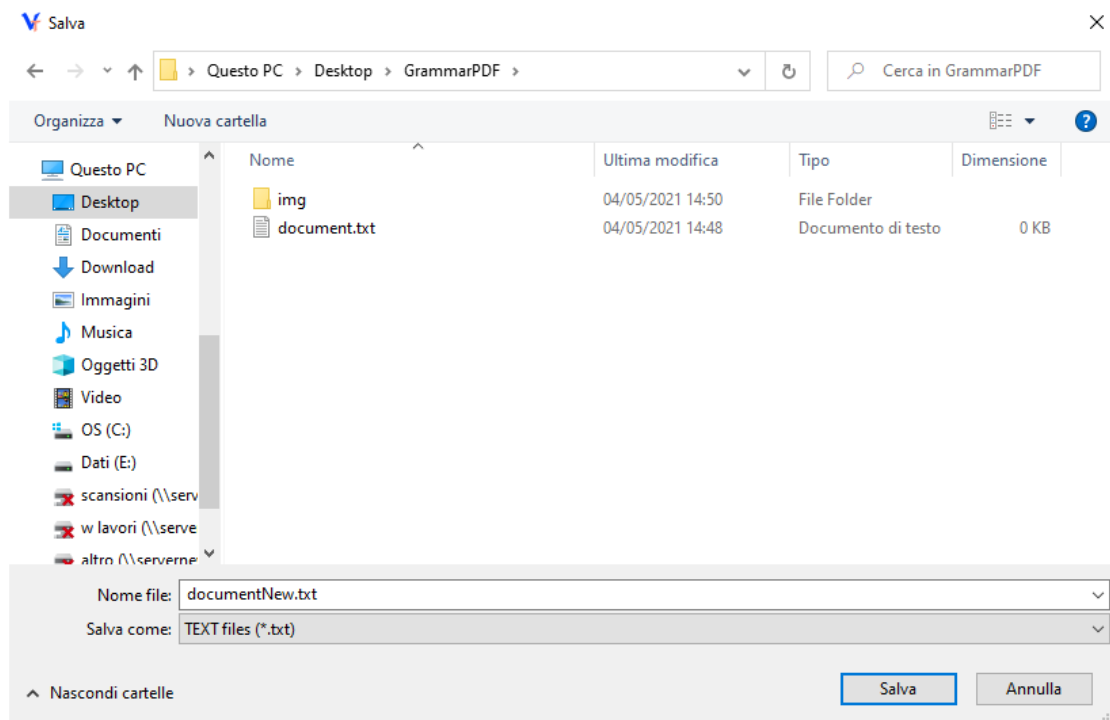
When the user is ready to start generating the document, it can be done by pressing the "Create" button and, at the end of the procedure, if it is successful, a confirmation message is shown, in addiction to any warnings that need to be reported to the user.

4.3. Reliability

The application was developed to tolerate incorrect usage by the user without compromising the execution of the software, but instead showing clear error messages that can help the user to understand the cause.

For example, if the user, while opening a document, enters the name of a non-existent file, in the first version, he returned the message "The specified file cannot be found" in the textual error area, while currently the The system explorer helps to check the existence of the document. Another

example is the situation where the destination path of the pdf file already exists and is used by another process; In this case, the message "Cannot access the file. The file is being used by another process".

C:\Users\stefa\Desktop\GrammarPDF\Test.pdf (Impossibile accedere al file. Il file è utilizzato da un altro processo)

A possible example of grammar is the following one, which describes a simple PDF document called Test containing a single page of 100x100 size, two coloured containers and a rotated text:

```
{
        title:"Test";
        page:{
                p_width:100;
                p_height:100;
                div:{
                        id:"A";
                        shape:"CIRCLE";
                        color:#482A4822;
                        width:50mm;
                        height:50mm;
                        pos-x:100;
                        pos-y:100;
                        position:cc;
                        fit-x:true;
                        angle-rotation:56;
                }
                div:{
                        id:"B";
                        position:lc;
                        color:#58184544;
                        width:50mm;
                        height:50mm;
                        pos-x:60%;
                        pos-y:60%;
                        fit-y:true;
                }
                text:{
                                id:"TESTO2";
                                colorT:"black";
                                position:ct;
                                height:30mm;
                                string:"Prova";
                                angle-rotation: 45;
                }
        }
}
```
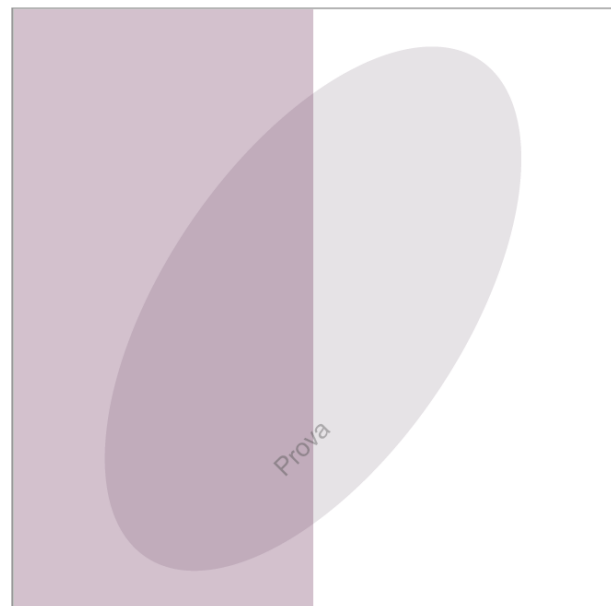
It is interesting to note that the application generates a PDF document even with the presence of errors in the grammar, however, returning a warning message to the user.

For example, by eliminating the semicolon after 'p_width: 100' in the previous grammar, the software produces a correct document, but reporting the error through the graphical interface.

```
line 5:2 missing ';' at 'p_height'
PDF GENERATO!
```

5. Possible future developments

During the development of the application, some interesting possible additions emerged for the expansion of the project. Here are exposed some ideas:

- The *underline* attribute, even if implemented, does not work correctly due to effective methods in the PDFBox library and incompatibility between PDFBox and PDFBox-layout. The generation of a function that allows to underline text with a more simple mode in terms of programming and calculation is in future plans.
- The implementation of constructs such as tables, links, mathematical formulas and graphs.
- A possible improvement in grammar such as simplification of constructs, id without double quotes, colour management also through other formats.
- Code optimization, especially for the VolTextListener.java class.
- Management of containers within other containers, management of objects internal to non-rectangular containers, management inside the grammar of particular constructs to reduce the document drafted by the user.
- Implementation of auxiliary tools for previewing the document before the generation.