GLASS Design Document

Input/Outputs:
The entire GLASS architecture starts with the input of a source file and a syntax specification file. The syntax specification file is a user-specified file that consists of the token expressions and grammar productions that define a language syntax. Users can create these files by hand themselves or use our GUI tool to define one. GLASS has multiple outputs throughout the process which are chained together to build the process model shown in Fig. 4. The idea here is that a user may want to only use certain parts of the system and stop the process there. These intermediary checkpoints include the following:

- The user can input a syntax specification file and the program will determine if it is properly formatted
- The user can input a syntax specification file along with a source code file and the program will determine if the source code can be parsed using the rules in the syntax specification.
- The user can input a syntax specification file along with a source code file and, assuming the source code file is written according to the rules of the syntax specification, the program will output a "XML-ized" version of the original source code file.

Lexer Module:
The primary purpose of the lexer module is to break down the input file into a token stream able to be comprehended by the parser generator. The input is both the input file and the syntax specification file. The AddTokenRule and AddIgnoreRule are primary methods of the token module, taken from the syntax specification file to tell the Lexer what is and isn't a token consumed by the grammar (for example, comments are allowed tokens but are not consumed by the language grammar, and would therefore be ignored tokens). Another primary method of the Lexer is the lex method which drives the entire lexical analysis operation once the tokens have been decided.

Parser Generator Module:
The primary purpose of the parser generator module is to create a parse tree using the token stream and the grammar productions defined in the syntax specification file. The parser generator uses the LR(1) parsing algorithm to parse the token stream based on the defined grammar productions. The parser generator module outputs a parse tree data structure which is to be passed to the XML Converter.

GUI Module:
The GUI module is an optional user interface that takes user input to generate a syntax specification file used for our parser generator. The base screen shown in Figure 1 is used to show the overview of the current grammar in a top-down form. The two right buttons are used to add new expressions and components. Figure 2 demonstrates what will appear when a new expression is to be added. Once a user is finished defining their grammar they can export it to a file which can be used as input in our parser generator.

Figure 1: GUI PyQT Mock Up

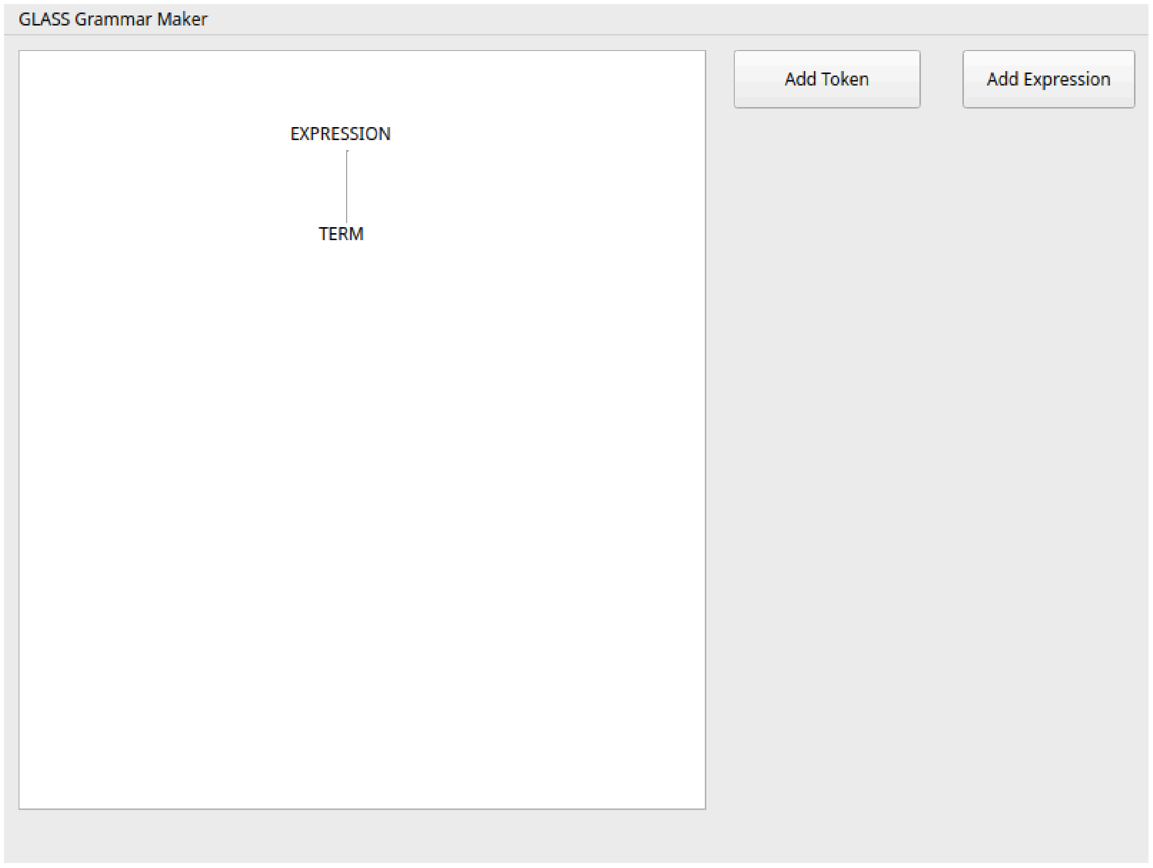
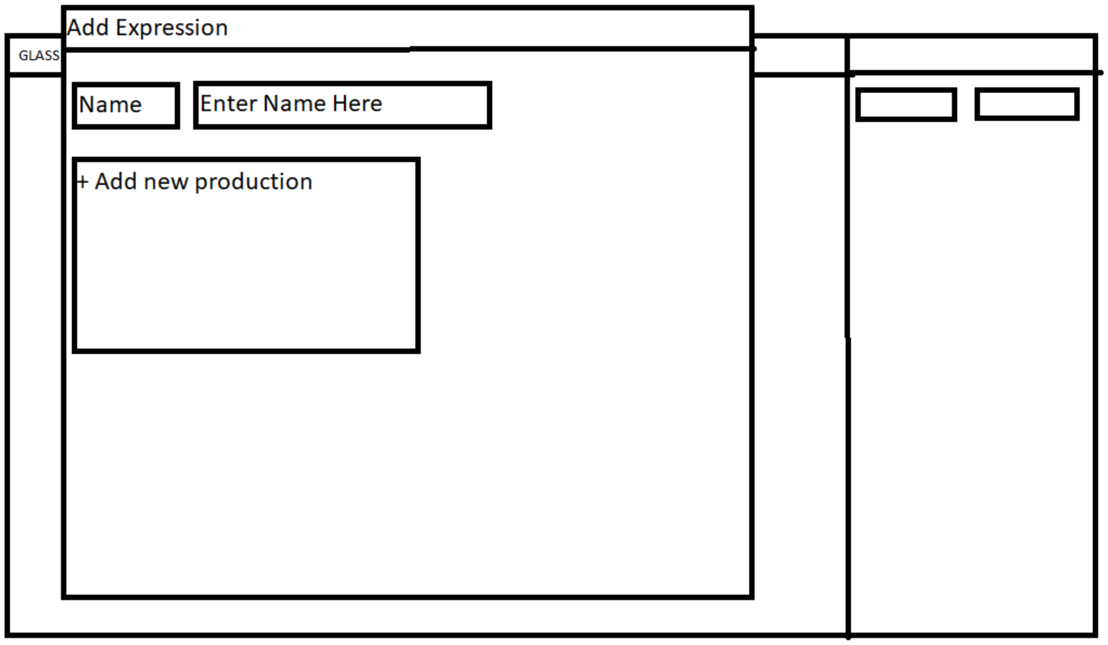


Figure 2: Add Expression Menu Mock Up



Parse Tree XML Converter:
The primary purpose of this module is to create an XML file which is a copy of the source input file with optional tags placed based on the parse tree. The input for this module will be the source file and the parse tree and it will be called in our process flow (see Figure 4). The module follows a node visitor design pattern with a primary driving visit function which performs a depth-first search of the parse tree checking to see if the node it is on has an XML tag associated with it. The module tags all applicable nodes inside of a newly generated XML file. The XML file is written to the system locally and the idea is that the XML file with all tags removed will be identical to the source file, which means the source code is fully preserved.

Macro System:
This module is an optional application of our GLASS tool used to modify the XML produced by our process flow (see Figure 4). The macro system consists of a base "AbstractMacro" class in which all macros are derived (see Figure 3). The reason this is done is that we would like for advanced users to extend this to write their macros. The primary method here is accomplished using the query function which uses a user-defined XPath query to find a certain tag within the XML tree. In our UML diagram you can see a FindReplaceMacro has already been defined which is used to replace text within a tag specified this is a simple example implementation of the base AbstractMacro class and is used by a FindReplaceMacroController which is responsible for the execution of the entire macro sequence. The

architecture is defined in a way to make this system easily user-extensible. The idea is that GLASS will ship with several more predefined macros.

Figure 3: Macro System UML Diagram

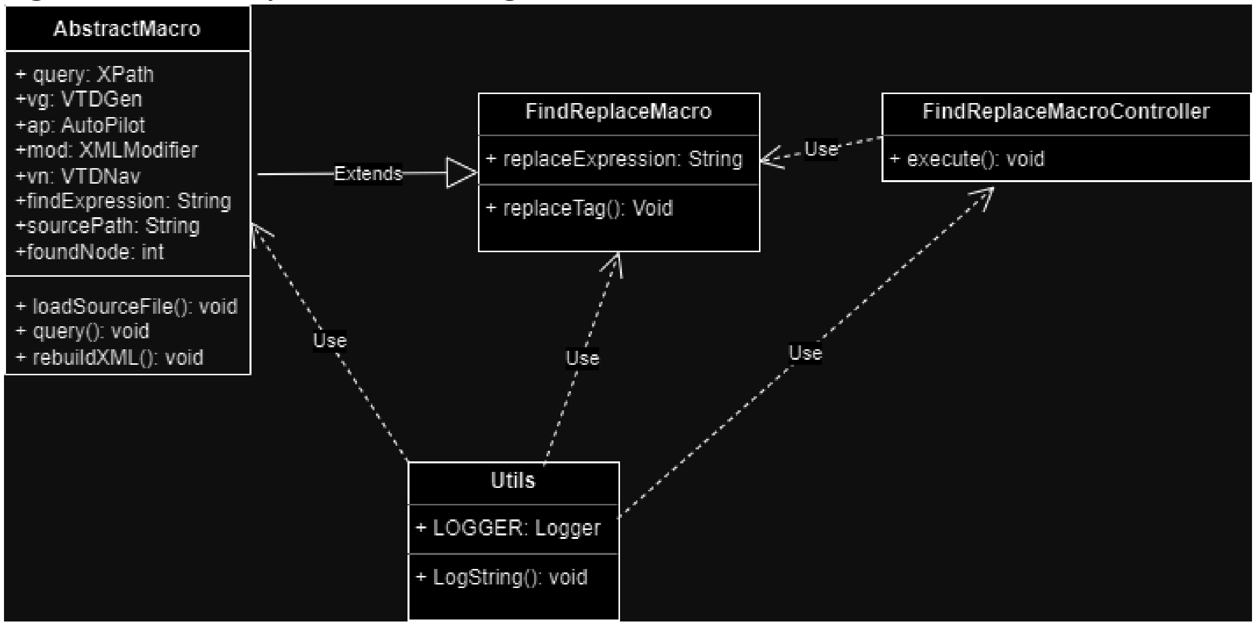
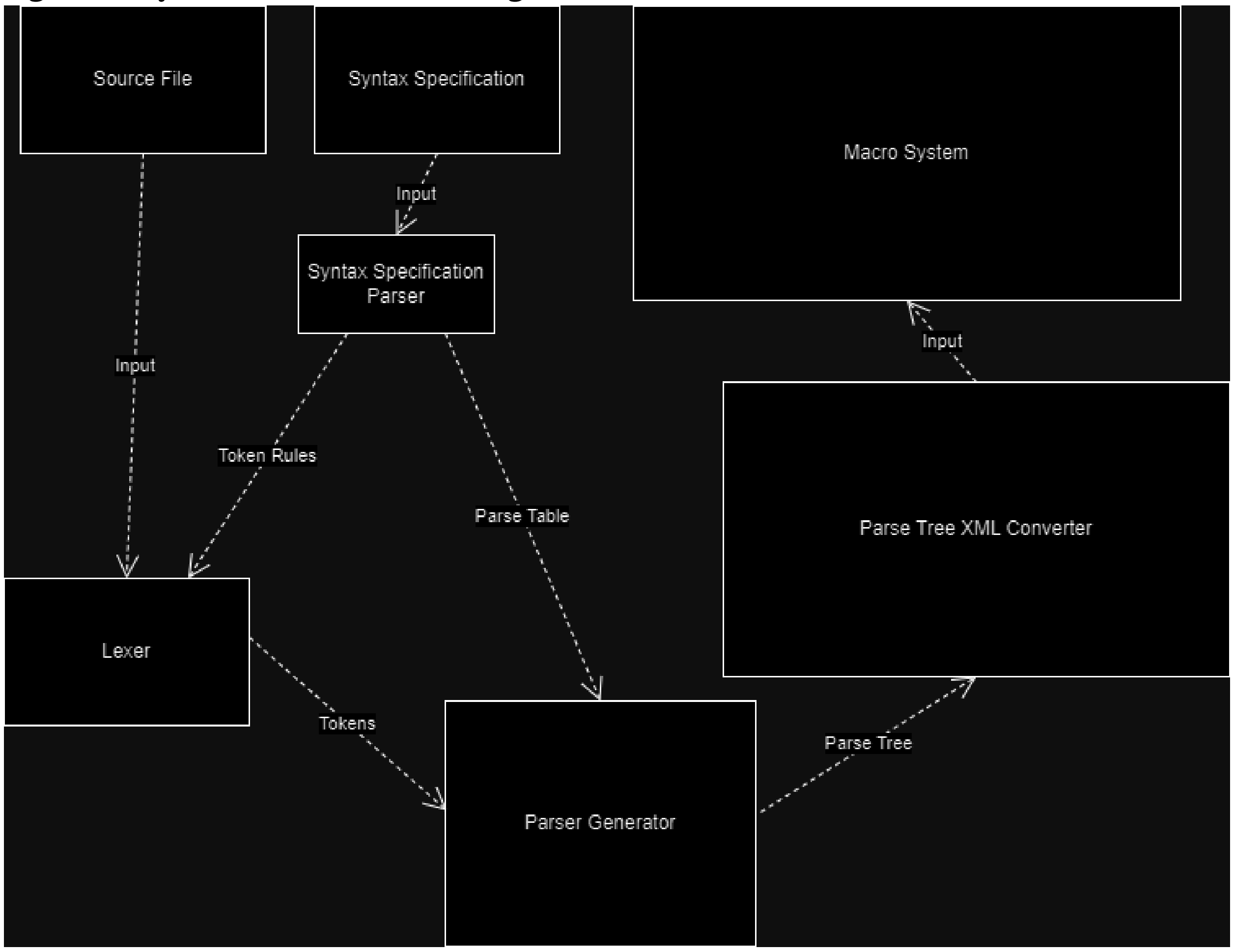


Figure 4: System Architecture Diagram


Source File
Syntax Specification
Macro System
Input
Syntax Specification Parser
Input
Input
Token Rules
Parse Table
Parse Tree XML Converter
Lexer
Tokens
Parse Tree
Parser Generator