Design Document for the comments and indentation preservation

Sbastien Combel

February 21, 2011

Contents

1 Introduction

The aim of this project is to support Modelica code refactoring with the minimal disruption of user-defined comments and indentation. Currently, only the declaration comments are handled. The C-like comment and user-defined indentation are discarded or moved.

2 Context / Overview

The lexer and the parser are generated by ANTLR3. All the generated code must be done in C (no C++).

The lexer convert a stream of character into a stream of tokens. The parser then convert that stream of token into an Abstract Syntax Tree (AST). Currently, only an AST is generated by the parser.

The refactoring operations use the AST (or a similar intermediate representation) to calculate the modifications to be done on the source code. Those modifications are stored as a list of new AST nodes, with AST position to unparse and buffer (or file) positions. Once the modifications are known, the specific part of the AST involved is unparsed, the modifications are applied on the buffer. Finally, the code is parsed again to update the data about the current source code.

3 Steps

First, the informations about the tokens must be stored in a seperate list, during the parsing of the source code.

Once it's done, the list will be used to generate a hashtable with MetaModelica code.

During refactoring (and/or unparsing ?) whenever the source code will have to be modified, the compiler will refer to the hashtable to know where the modification must be done. A modification can only occur within a token string, to avoid disruption of the surrounding code.

4 Idea to keep

Refactoring Down-top way, in order to not disrupt the coordinates of the next nodes.

5 Confused things/Problems

A refactoring example is the renaming of a component. Aim of the function, renaming a component at its declaration and calls. Since the modification generated by this example will take place in several places in the code, I suspect

that an ordered list of those modifications must be done before (or else the down-top way will be a waste).

Q :

Once the refactoring has taken place, shall the hastable be regenerated from scratch, or shall we try to update it ?

If multiples refactoring ¡¡ functions ¿¿ are called in the same ¡¡ step ¿¿, shall we try to apply all of them at the same time (i.e. calculate everything and sort all of the modification before appliance), or shall we execute them one by one, updating/regenerating the hashtable between each function ?