Modelica Parser and OpenModelica AST-builder

Contact: Peter Fritzson (petfr@ida.liu.se, tel: 0708-281484)
or Adrian Pop (adrpo@ida.liu.se)
PELAB – Programming Environment Lab, Institutionen för Datavetenskap
www.openmodelica.org

At PELAB, together with the Open Source Modelica Consortium (an international open source effort supported by 28 organizations, see www.openmodelica.org) the OpenModelica environment including the OpenModelica Compiler (OMC) of the Modelica language including MetaModelica extensions is developed. The development is open source under the OSMC-PL and GNU V3 licenses.

Currently an ANTLR-generated parser is used that has several drawbacks: execution and memory overhead due to the fact the an ANTLR syntax tree is first created which is converted into the internal MetaModelica AST representation (via an intermediate C-based tree node representation. The current parser also has quite bad error handling, which is important for novice users of Modelica in the OpenModelica environment. It is also hard to maintain and extend due to the complexity of the ANTLR environment.

The goal of this master thesis project is to create a more efficient parser and AST-builder, with good error handling, using the existing LL(1) Modelica+MetaModelica grammar as a starting point/raw material. The development environment is Eclipse, using the MDT (Modelica Development Tooling) Eclipse plug-in and debugger.

There are basically two options:

1. Write a recursive-descent parser in MetaModelica based on the current LL(1)/EBNF grammar. This has the advantage that the MetaModelica AST constructors can be called directly (i.e. efficient), good error handling can be done since a top-down parser knows about the current parsing context, and avoidance of certain low-level programming pointer errors that can occur if you do a mistake in applying AST-building C-macros. There is already a small recursive-descent parser and scanner available for a small subset. The disadvantage is that you have to write a recursive-descent parser by hand, which on the other hand is an advantage when implementing good error handling.

2. Transform the grammar into a form that fits an LR-parser generating tool such as Bison, together with Flex. Generate a parser that builds Meta-Modelica AST trees using AST-building C-macros. This has the advantage of using a parser generator, but the disadvantage of the possibility of hard-to-find bugs if you do a small mistake in using the C-macros. Also, bison-parsers usually do not have very good error handling. It is also slightly less efficient since the nodes created by the C-macros must later be copied during MetaModelica garbage collection.