# MPLc Documentation

Tomi Karlstedt & Jari-Matti Mäkelä

July 11, 2008

# Contents

Chapter 1

# Introduction

MPLc is a MPL-to-AspectJ compiler. MPLc uses Antlr v3 to lex and parse MPL files and hand-written tree walking to transform the Antlr generated Abstract Syntax Trees to a printable AspectJ form. Running the MPLc program always prints an aspect if errors do not occur during the parsing or tree walking. Printing can be done into a file or into standard output (System.out).

MPLc version 1.1 is compatible with the MPL version 0.7.0 Antlr grammar. MPLc does not require Antlr to be run as the Antlr generated lexer and parser are provided in the JAR package. However, MPLc still depends on the Antlr runtime package.

In chapter 2 we describe the compile process. The chapter 3 goes through the implementation class by class. Chapters 4 and 5 give system requirements, installation instructions, and usage instructions.

# Chapter 2

# Implementation

MPLc is run by calling the main method of the MPLc class. The general "control flow" of MPLc can be described in 3 steps. The first step reads from policy files and turns the token streams into Abstract Syntax Trees. The second step manipulates the generated trees checking for possible errors and producing trees that are easily printed. This data is saved into a SemanticResult container. The final step is the actual printing process.

1. *"First pass":*

   a. The main method creates a new instance of FirstPassHandler which runs the `compile(String)` method for the given policy name. The `compile(String)` method first executes the Antlr-generated lexer and parser for the policy file which produce an AST.

   b. The AST nodes are separated into names, imports, policy variables, package imports, and rules.

   c. For each of the listed imports the method calls `compile(String)` method recursively and creates a new Module instance with references to the original class members. The pass keeps track of cyclic policy imports and issues an error if one is found.

   d. The pass produces a tree structure of modules which contains lists of all the defined rules, variables, and imports in all of the policies.

2. *"Second pass":*

   a. Trees containing class imports are transformed into list of Strings.

b. Policy variables are converted from trees to PolicyVariable objects. This is done recursively to preserve the introduction order from imported policies.

c. The variables are finally checked of type inconsistencies and the final default value is determined.

d. The rules are mapped into rule definitions, condition trees, and effect trees. The mapping preserves the recursive introduction order.

e. Finally the effects are validated to the policy variable assignment form.

*3. Pretty-print:*

MPLc creates a printer object with the set of data to be printed and the print stream. The aspect is printed through the `printAspect()` method. The printing process uses some tree rewrite rules with imaginary tokens to identify MPL construct related tokens. Apart from these rewrites the printing is straight-forward token printing from the given set of data.

In MPLc version 1.1 the AspectJ printer is still responsible of certain MPL-to-AspectJ transformation which may cause non-printing related error messages in printing phase. See Section 3.2 for more information.

# Chapter 3

# Structure

MPLc 1.1 includes Java files as follows:

- package: `mpl`
  - *MPLc.java:*
    The main compiler class which includes the main program to be run. See Section 3.1 for more information.
  - *MPLSet.java:*
    A definition for the MPL set variable used in aspects. Extends generic `HashSet<T>` by implementing the different set expressions.
  - *MPLShutdown.java:*
    An interface for the shutdown sequence. As of MPL 1.0 contains only the shutdown() method that must be implemented by the concrete shutdown object.
  - *MPLSimpleShutdown.java:*
    Gives a simple implementation for the shutdown sequence by terminating the program and giving an error message in System.err.
- package: `mpl.backend`
  - *AspectJPrinter.java:*
    The AspectJ pretty-printer class for the MPLc compiler. In MPLc version 1.1 handles some further MPL-to-AspectJ transformation which is described in Section 3.2.
  - *AspectCPPPrinter.java:*
    The AspectC++ pretty-printer class for the MPLc compiler. Not currently implemented.

  - *AspectPrinter.java:*
    Generic interface for all pretty-printer classes.

- package: `mpl.frontend`

  - *FirstPassHandler.java:*
    Loads imports and classifies AST nodes.
  - *FirstPassHandlerCore.java:*
    Abstract helper routines for the first pass.
  - *Module.java:*
    Represents the data of one MPL policy file.
  - *MPLLexer.java:*
    Antlr-generated lexer that takes an MPL policy file as input.
  - *MPLParser.java:*
    Antlr-generated parser that takes a token stream from the lexer as input and generates an AST for the given MPL policy. The form of the AST is defined in the MPL grammar for Antlr v3 (MPL.g).
  - *Rule.java:*
    Container for preprocessed lists of AST nodes of conditions, effects, and rules.
  - *SemanticPassHandler.java:*
    Processes all imports, java package imports, variables, rules, and effects found in MPL files. Also validates effects and variable definitions.
  - *SemanticPassHandlerCore.java:*
    Abstract helper routines for the second pass.
  - *SemanticResult.java:*
    Container for all relevant data gathered during the first two passes. This container is used to print the aspect.
  - *TokenTypes.java:*
    Token enumeration for AST manipulation. Automatically generated.

- package: `mpl.frontend.types`
  *Type*, *TypeList*, *TypeSet*, and *TypeSimple* represent all supported policy variable types.

- package: `mpl.frontend.variabl es`
  *PolicyVariable*, *VarList*, *VarPattern*, *VarPrimitive*, and *VarSet* represent supported policy variables.

## 3.1 Class MPLc

MPLc includes the main method that handles the given command-line options and starts the compiling process. In delegates the manual tree walking needed to transform the given input files into printable trees first to FirstPassHandler, which imports all required policy files, lexes and parses the input files, but also classifies the abstract syntax tree on high level.

A simple semantic pass follows the first pass. The second pass extracts more data from the syntax tree into easily accessible lists and partly validates the language (effects and variable conflicts).

Finally the main method chooses between available pretty-printers and output streams based on the given command line options.

## 3.2 Class AspectJPrinter

AspectJPrinter prints an aspect from the set of trees (SemanticResult) given as parameter in the constructor. The overall form of the aspect is generated in the printAspect() method which calls other methods to print more fine-grained constructs of the aspect including the aspect variables, pointcut definitions, and the different advices. Antlr rewrite rules with imaginary tokens are used to identify MPL-related constructs and modified accordingly while Java expressions are printed token by token.

Due to limitations in MPLc design and implementation the compiler object does not create a rule-specific variable type table. This leads to type checking problems in the printer implementation. In particular the printer checks element-to-set (e2s) conversion validity. Therefore it is possible that the printer object halts the compiling process with a proper error message. This will happen after the printer has written some amount of information into the defined print stream.

# Chapter 4

# Installation

MPLc is distributed in a JAR package containing the compiler and all of the required classes for using MPLc generated aspects. In order to generate aspects from policies you will need Java 1.5 or later and Antlr v3 or later. In order to use the generated aspects you will need AspectJ 5 or later. Also either mplc-rt-1.1.jar or mplc-1.1.jar is needed to compile the AspectJ aspects.

**Installation:**

1 Save `mplc-1.1.jar` and `antlr-runtime-3.0.x.jar` into a directory.

2 Add the saved `mplc-1.1.jar` and `antlr-runtime-3.0.x.jar` to the CLASSPATH environmental variable.

**Requirements:**

- Java 1.5, Antlr runtime

**Optional requirements:**

- AspectJ 5

# Chapter 5

# Using MPLc

MPLc is used from the command line like:

```
java MPLc <options> <policymodule>
```

Where policy module is the name of the transformed policy. The policy must be found in a file called `<policymodule>.mpl`. In MPLc 1.1 the extension doesn't need to be omitted. If a policy includes further imported policies they must be found in separate .mpl files either in the current directory or in same directory as `<policymodule>.mpl`.

MPLc creates a file called `<policymodule>.java` (unless other options are used) to the current directory where `<policymodule>` is the name of the given policy.

MPLc options include:

**-print** Prints the aspect into System.out instead of a file.

**-shutdown <name>** Uses an object of type `<name>` as the shutdown sequence. The class `<name>` must implement the interface MPLShutdown

**-aspect <name>** Prints the aspect into a file called `<name>.aj` instead of `<policymodule>.aj`.

**-language <language>** Generate the aspect code in given language, supported options: `cpp` (AspectC++) and `java` (AspectJ).

**-debuginfo** Generates extra runtime code to the aspects that prints debug info whenever effects are being executed or conditions tested.

To compile the generated aspects, Java 1.5 compliance must be used with the AspectJ compiler as well as the `mplc-1.1.jar` or `mplc-rt-1.1.jar` must be found in the CLASSPATH env variable.