Planeringsrapport

Domain-specific language for high-level sampling tasks in high-performance computing

Martin Hardselius Viktor Almqvist

2 februari 2012

1 Background

2 Aim

3 Problem

4 Scope

An important part for us is to get a working solution since our result supposed to be used by the users of the system. The system input is currently in XML form, but our compiler could render using XML redundant. But as we mainly want a fully functional compiler, we will generate XML code. This way we will not need to study or re-write some of the code to change the input data of the system, excluding our compiler. We will add Python code generation if we have time to continue develop the solution, which is a solution without the unnecessary XML code generation step.

5 Method

Our method for making the compiler is devided into five steps; study input data, language syntax, parser, type checker and code generation. The code generation part includes the additional optional part describe in the scope section above. The last three parts are simply the standard three parts of a compiler, while the first two are part of designing the language to suit the users.

We have not included an optimization step in the compiler due to the nature of the language. It is a language to describe dataflow networks, but not to execute any code segments. We may include some sort of optimization when we understand more of the project.

We have chosen to write the compiler in Java instead of in Haskell, which we are more knowledgeable in. There are two reasons for this decision. The first is to ensure that more people can take over and understand our code when we leave the project, since Java is a more commonly known language than Haskell. The second reason is that we want to learn and become more adept Java coders before we graduate from Chalmers.

5.1 Research

Our first objective is to fully understand the structure of the input data. We will studying project code, the XML code examples used for testing the system and discuss the design choices with the developers. We need to understand how the input data hierarchy works, functions and subnets are defined, etc. Understanding these structures are necessary to continue to the next part of the method.

5.2 Design

After and probably during the study period, we will start defining a language syntax. We expect to make a simple draft to show our ideas for a structure of the

language. We do not want to define the whole syntax by ourselves since we may miss important features which should be included. That is why we will show the draft to the developers to get their input and own design ideas. Working from this point with a continuous input from the developers will result in us creating a good syntax that they are satisfied with and we, as language creators, think is a clean syntax and know will be a simple and effective to implement.

5.3 Implementation

5.3.1 Parser

We can start implementing the compiler when the language syntax has been established. The first step is to write a parser for building abstract syntax trees. We will use BNFC which is a tool for generating a finished parser from BNF grammar. Our objective is to write a BNF grammar which represent our syntax defined in the previous step. The grammar will contain the defenition of statements, expressions, types, etc.

5.3.2 Type Checker

Some sort of type checking will be implemented to make our contribution a more powerfull tool for the users. It is hard to know at this point what kind of incorrect code is possible for a type checker to do to find, since we do not know anything about the language.

5.3.3 Code Generation

Once we have a type checker we can start with the final part of the compiler, the code generator. The code generator will generate XML code which can be used directly as input for the current system without any complications.

The optional additional part is to skip the XML code generation and generate Python code directly. As we understand it, the system does currently convert the XML code to Python code which makes this part not necessary but a better solution. If we would add this part we would have to re-write some of the current system to take Python code as input instead. By doing this step we would eliminate the redundant step of generating XML code, but it will also take more time.

6 Time Plan