1. Introduction to the problem

The topic of this project proposal is positioned in the area of Programming Language Theory. It triggers both the sub-fields of Compilers and Modelling Languages.

Although this topic mostly refers to Computer Science, it is also closely related to Control and System Engineering. Indeed, purpose of Modelica language and tools is to help design and simulation of complex systems.

Modelica is a domain specific language used to describe systems trough (algebraic or differential) equations. It is a non-proprietary object-oriented language. Modelica tools can be open source or commercial and often include whole environments for simulation and analysis, not only a compiler.

However, Modelica compilers are the core of these tools. Their distinctive feature is that they need to translate from a language that is descriptive to another that is prescriptive (e.g. C, C++ or machine code).

Compiled Modelica code loses its object-oriented structure because of flattening, which is the process of lowering structured models and components into a large model with no depth. On one hand, flattening permits a less complicated algorithm for computing the solution of DAE\(^1\) systems. On the other hand, it causes an increase in its time and space complexity which may cause the impossibility to simulate too large models\(^2\).

Modelica is nowadays used in research, both industrial and academic. It is domain neutral, hence it can be practical for a wide range of problems. Examples of industries using Modelica are automotive companies\(^3\) and power plant providers\(^4\).

Any improvement in Modelica environments will affect the research in other fields. Examples include, but are not limited to, the above mentioned.

The problem to be solved is that current compilers do not scale well with the size of the systems due to flattening, which translates into very long time to simulate large models and the impossibility to compile over a certain threshold due to lack of computational resources \(^3\).

The idea to improve performances is to postpone flattening as much as possible, maintaining the object-oriented structure of the models as far as possible. In particular, the main goal is to avoid loop unrolling and make use of the vectorial equations present in the code. To achieve this, new algorithms have to be designed while some already existing have to be adapted in order to preserve array structures.

Speeding up the process of compilation would have effects on the time-to-solution because Modelica models are supposed to be simulated only once to obtain the results. Any other run with different input includes re-compilation. Hence, compilation time is as important as run time for this specific field.

Improvements on space complexity would have an effect on time complexity too (a large file needs in fact a long time to be compiled).

---

1 Differential Algebraic Equations
2 They may need too much time or they may even fail compilation due to lack of resources, e.g. RAM \(^1\)\(^3\).
3 For example, Audi, BMW, Daimler, Ford, Toyota, VW.
4 For example, ABB, EDF, Siemens.
2. Main related works

There are several commercial tools on the market but, since they have proprietary licenses, their code and algorithms are not disclosed. Some of them are Dymola [6], AmeSIM, CyModelica.

OpenModelica[5] is the most famous and relevant work related to Modelica simulation. A research activity involving OpenModelica is multi-core parallel code generation [7]. Even though OpenModelica is not the most performing tool on the market, it represents the most relevant open source effort in the Modelica community so far.

An idea to rewrite from scratch a Modelica compiler with the main goal of high performances is shown in our research group’s recent work [1]. The new compiler would use the LLVM [5] compiler framework and skip the intermediate C or C++ code, generating an intermediate representation that retains structural information from the original model and producing machine code that is optimised for the target hardware architecture. The main goal of the research is to preserve loops and arrays. The idea is to use an object for the function to compute the residuals of DAE, which has to be generated only once and then called using different inputs and outputs each time it is needed.

In previous work [3] are discussed in more detail ideas for a prototype of an optimised compiler and is shown, as an example, the case of a thermal conduction model compiled in OpenModelica and Dymola, against a hand-written C++ code. The results show that the gain of the time-to-solution of the C++ code grows exponentially. The results of the prototype code are due to the absence of the flattening phase with results in a model which is smaller in size (as it maintains its hierarchical structure). Another issue discussed [3] is that each equation is mapped into a C function, resulting in a larger code size and in overhead due to function calls. Together with the loop unrolling used in the existing compilers, this causes the presence of a host of functions that are essentially the same function with different arguments.

3. Research plan

The goal of the research presented in this document is to contribute to the implementation of a Modelica compiler that generates simulation code that is better than that of the state-of-the-art tools. This will be achieved by preserving (as much as possible) data structures in the models, primarily arrays, in order to avoid scalarization and obtain performances that scale well with the size of the model.

The nature of the research is both theory and implementation, since it will include the study of new (and modification of existing) algorithms and their application in a real compiler. Therefore, the product of the research will be software before publications.

The tasks in which the research can be decomposed are:

1. **State of the art**: this task includes the study of the literature about Modelica tools and their current performances and techniques, as well as compiler technologies that can be applied to the case.

2. **Benchmarking**: this task consist of the detection of significant use cases that exploit vectors and loops, their implementation in Modelica and testing of the performances.

3. **Analysis**: the results obtained from the benchmark of the previous task will then be evaluated in order to find where and why the performances are not satisfactory.

4. **Implementation**: in this phase, new algorithms will be designed or, when suitable, existing ones will be adapted to make use of the preserved data structures. These algorithms will then be implemented.

5. **Evaluation**: the features implemented in the previous task will be tested for correctness. In case of failure, the implementation task will be resumed to fix the mistakes found. Once the software passes the correctness trial, its performances will be evaluated in order to prove whether there is an improvement from the existing tools.

---

5[www.openmodelica.org]
6. **Writing:** the last task of the project will be the composition of a conference paper describing the results achieved.

Figure 1 shows a Gantt diagram for the tasks. The **state-of-the-art** phase is already in progress. The **implementation** and **evaluation** tasks will be carried on partially in parallel, since the products of the first will be tested as soon as ready, in order to correct any error as it arises.

![Gantt Diagram]

The metrics that will be used to evaluate the output of this project will be time and space complexity, as for any other software.

The processes that will be evaluated will be both the compilation and the simulation, because they always are performed together, using new and existing benchmark suites [2, 4].

**References**


