4 Interoperability of Languages with Generics, using Aldor, C++ and Java

Yannis Chicha, Florence Defaix & Stephen Watt
University of Western Ontario

Aldor is a language offering functional concepts and parametric polymorphism. At the same time, this language has been specially designed to fulfil the needs of the computer algebra community, with a type system expressive enough for mathematics, and admitting a high degree of optimisation. So should everybody forget about C++ and begin to use Aldor? Well, no — the huge base of existing C++ libraries will never be re-implemented in Aldor. We therefore consider interoperability between C++ and Aldor. This could be achieved either with a loose coupling (e.g. interprocess communication, CORBA architecture), or with a tight coupling (direct function calls in the same address space). We consider here the tight coupling model.

Calling C++ from Aldor

To be able to use C++ code in Aldor, we have explored potential relations between the object models of both languages. C++ is a class-based object-oriented language. It provides the following idioms: classes, abstract classes, virtual methods and generic programming through static templates. Aldor is a functional language offering higher-order functions of dependent type. It provides a two-level object model. Type categories are like contracts: if a type belongs to a category, we have a guarantee (checked at compile time) that this type actually implements the functions listed in this category. Domains (type) belong to these categories and actually implement the functions they list. Categories and Domains can be created by functions. The parameterisation of such functions provides generic programming.

To use C++ code in Aldor, we establish a correspondence between the main entities of both languages. It is obvious that we won't be able to establish an exact correspondence for every concept in the two languages. Our goal is to provide a rich enough interface to allow most of the code to be interfaced in a reasonable way. Our tool is *not* a C++ to Aldor translator. Rather, it generates interface codes so that both languages can work together. Bodies of C++ functions or methods are not taken into account.

To use C++ code in Aldor, it is not sufficient to "import functions", we want to provide a reasonable interface containing types that are generated from C++ classes. For each abstract class, we should generate a category. A class will correspond to a domain whose category exports all the public methods. Accessors will allow for manipulation of public variables. Template C++ classes become parameterised functions producing types in Aldor.

To let Aldor-generated functions call C++ code, we need some kind of link. It happens that both languages have a strong connection with C, and this fact is used by our interface.

An `XML` representation of a C++ code will allow us to generate easily and precisely our "stub" code. `XML` is very easy to parse and to manipulate. The tool divides the work into two steps: C++ to XML and XML to Aldor. Once the code is generated, Aldor programs using C++ types can be compiled.

In this talk, we present an overview of the global architecture and the model correspondence. A small example allows us to explain how to use the tool.

Calling Aldor from C++

We have built a second tool to make Aldor code callable from C++. This still requires the generation of entities corresponding to the idioms of the source language. The main issues are the object model and the

basic type correspondence. The object models we consider have already been studied for the tool to call C++ code in Aldor. An Aldor domain corresponds to a C++ class and an Aldor category is translated to a C++ abstract class.

Parameterisation constitutes a tough problem. Aldor parameterisation is much more powerful than C++ templates. Whereas C++ templates are completely static, it is important to be able to provide a reasonable translation of Aldor's domains producing functions which provide statically-checked dynamic parameterised types. We simulate a dynamic behaviour for template classes using C++ virtual functions.

To generate C++ code from Aldor code, we modified the Aldor compiler. A new option `-Fc++` generates necessary stub code to interface C++ with Aldor. The basic type correspondence has to be flexible enough to use different Aldor libraries with a C++ code. For example, the `basicmath` library provides a different definition for some Aldor types (`SingleInteger`, `String`, ...). The user is allowed to provide a file containing the exact correspondence between C++ types and Aldor types. This allows any correspondence (and thus any library) to be used.

Java

Java is another language for which an interoperability with Aldor is desirable. In addition to some properties such as compilation to a platform independent byte-code, Java has a broad standard library which would be an asset to use in Aldor. The third part of this work is a feasibility study of a Java/Aldor interface. We have started to examine the different possibilities to achieve a Java/Aldor interface. Two methods have been considered: compiling Aldor to Java byte code; or using the C interface provided in both languages. We present these two solutions and the pros and cons of each, along with some practical experiments we conducted.