Concept-based C++ template overload compilation with XML transformations

Szabolcs Payrits and István Zólyomi

Parametric polymorphism provides a high level of abstraction by enabling type arguments in function and type definitions. This programming paradigm is implemented by templates in C++. Type safety of parametric polymorphism is greatly improved by concept checking which allows formal specification of type parameter requirements. On one hand, concepts help application developers to implement correct parameter types conforming to static type-safety. On the other hand, concepts also make it possible to overload polymorphic functions based on selected characteristics of the parameter types.

Currently there is no established C++ standard for defining concepts, although several proposals compete to be adopted into the standard. The common approach in these proposals is to define concepts similarly to interfaces in other languages, e.g. Java. In this paper we suggest another possible alternative, similar to current type traits definitions. We assume a set of elementary predicates about C++ types and we define concepts as a composite logical expression of these elementary predicates.

Advanced C++ metaprogramming techniques also require a template overloading mechanism based on template type arguments. However, currently C++ template overloading is fundamentally based on the Substitution-Failure-Is-Not-An-Error (SFINAE) rule. This rule makes template overloading dependent on tracing of internal compiler behaviour, what makes template metaprogramming extremely vulnerable to compiler bugs and compiler-dependent relaxations of the standard.

In this paper we show that in selected cases it is possible to replace the current template overloading approach based on SFINAE with template overloads based on explicitly defined concepts. By eliminating SFINAE, we can separate the compilation of template-containing C++ programs into two phases. During the first compilation phase, resolution of template overloads based on our concept predicates and template instantiation is done. Second phase is the compilation of a c++ program containing no templates into binary code.

With this separation of phases, it is possible to use an independent meta-compiler tool for the execution of the first phase of the compilation. The second compilation phase can be done with any standard C++ compiler.

As a proof of concept, we have started to implement the-phase compilation mechanism. Currently, our C++ concepts are defined as C++ comments for template parameters. Compilation consists of three steps: in first step, we use the Columbus C++ parser to transform our concept-containing C++ program into an XML-based representation called CPPML. Template instantiation with concept-based template overloading is implemented as an XSLT transformation on the CPPML language, resulting in another CPPML document with all templates instantiated. In the third step we convert the transformed document back into standard (template-less) C++ code.

References

[1] B. Stroustrup. The C++ Programming Language (3rd Edition), *Addison-Wesley Professional*, June 1997.

[2] J. Siek, D. Gregor, R. Garcia, J. Willcock, J. Järvi, and A. Lumsdaine. Concept for C++0x, *Technical Report N1758=05-0018, ISO/IEC SC22/JTC1/WG21*, January 2005.

[3] G.D. Reis and B. Stroustrup. Specifying C++ concepts, *N1886, JTC1/SC22/WG21 C++ Standard Comittee*, October, 2005.

[4] F. Rudolf and Á. Beszédes. Data Exchange with the Columbus Schema for C++, In *Proceedings of the 6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pages 59-66, IEEE Computer Society, March 2002.