Using C++ Templates for Implementation of Patterns

Angelika Langer,
Consultant, Germany

Originally, templates were introduced to the C++ programming language in order to express common structure in form of parameterized types implemented as class templates. Function templates, in addition, allow expression of common functionality. Using the template language feature in C++, behavioral patterns as defined by Gamma et.al. can be implemented using templates instead of inheritance and virtual functions. Often, classic object oriented runtime polymorphism can be replaced by a compile time polymorphism, the effect of which is a considerable gain in runtime performance.

The template technique leaves behind classic object oriented approaches, and for that reason many C++ programmers do not even consider it as an alternative. Yet it has been widely used in areas of C++ programming, where performance is crucial: foundation libraries such as the Booch components and the STL prove that the template version of strategy and other patterns are of great value in practice. In this tutorial, the alternative template technique is demonstrated using the strategy pattern as an example. The intent is to expand the attendants’ set of programming techniques by another useful mechanism based on template programming. Naturally, the decision between runtime techniques on the one hand and compile time mechanisms on the other hand cannot be made without a sound knowledge of the respective trade-offs. Hence, we will carefully evaluate the respective advantages and drawbacks.

Angelika Langer works as a freelance trainer and mentor. Her main area of interest and expertise is object oriented software development in C++ and Java. She writes a column for the US magazine C++ Report, is a recognized speaker at OO conferences all over the world, contributed to the standardization of C++ on the ISO/ANSI committee, and is author of the book "The Standard I/OStreams and Locales". For further information see http://www.camelot.de/~langer/.