AspectC++ – Execution Model Overview

©2005 Olaf Spinczyk <os@aspectc.org>
Friedrich-Alexander University Erlangen-Nuremberg
Computer Science 4

May 20, 2005

This is an overview about the AspectC++ execution model prepared for the European AOSD Network of Excellence. Detailed information on AspectC++ including manuals and tutorial slides with lots of examples is available from *the AspectC++ Project Homepage*.

The overview is based on the documentation for AspectC++ 0.9.3. The execution model is still "work in progress".

Models

Architectural Characteristics

The following objectives of the AspectC++ development effort are responsible for the design of the AspectC++ execution model:

- The resource consumption of AspectC++ applications should be minimal. Compared to a tangled implementation of the same functionality there should be *no overhead*.
- AspectC++ should fit into the "philosophy" of C++, i.e. as much as possible is done at compile-time.
- AspectC++ should not rely on any runtime infrastructure provided by C++ such as a heap or RTTI. These features are often not available in embedded systems projects or system software.

AspectC++ is implemented by a static source to source transformation from AspectC++ to C++. A version that is able to generate pure C code if the component code language is C will be available in the near future.

A separate AOP run-time infrastructure is not needed by the generated applications. A small number of helper functions, classes, and templates is woven directly into the application code modules.

Aspect Model

The AspectC++ weaver transforms aspects into ordinary C++ classes. The generated aspect classes do not inherited from a specific (common) base class. Therefore, the object layout is not affected by the tranformation.

Advice Model

After the transformation, advice becomes an aspect (class) member function. If the system is compiled with optimizations enabled, the advice code will typically be inlined into the component code. The argument list of the advice (function) contains the context variables that are passed from the join point to the advice and an optional `JoinPoint *tjp` pointer. This pointer is only passed if the advice code actually uses `tjp`.

In the case that the advice uses `tjp`, static members, or types defined in `JoinPoint` the advice is transformed into a template function with `JoinPoint` as a template parameter type. Due to the template instantiation mechanism of C++, multiple instances of the advice code might exist at runtime.

Pointcut Model

Pointcuts have no runtime representation.

Functionality

Joinpoint Shadow Retrieval

In the static AspectC++ weaver `ac++` , joinpoints are retrieved only at compile time. A complete syntactical and semantical analysis has to be performed on the AspectC++ code. The result is a syntax tree representation with links to symbol tables, which in include the results of the semantic analysis, and links to tokens in the input file. The token links are needed for the code transformation. Based on the symbol tables the set of static joinpoints (see language description), execution joinpoints, construction joinpoints, and destruction joinpoints is constructed. A visitor is used to find all call joinpoint shadows in the syntax tree.

Weaving Approach

Our weaver `ac++` is usually applied directly before calling the C++ compiler. With the `ag++` front-end both steps are even integrated into one tool. Depending on the advice defined in the program, the weaver performs the following manipulations:

- function calls are replaced with calls to wrapper functions that call the advice chain
- function implementations are replaced with wrapper functions

- new members are inserted into classes
- `#include` directives are generated
- `#include` directives are expanded (only in the so-called "single translation unit" mode)
- aspects are transformed into classes
- the `aspectof()` function is generated if there is no user defined one
- code for the instantiation of singleton aspects is generated
- code for cflow management is generated (enter, exit, check, state instance)
- helper functions for runtime type condition checks on objects are generated (for `that`, `target`, and combinations)

Currently, all aspects of the project become “friends” of all classes and other aspects of the project. A more sophisticated access control mechanism is future work.

Introduced members become ordinary members. Their visibility (private, public, or protected) is taken from the visibility of the advice declaration in the aspect body.

Advice code and the generated wrapper functions, which call the advice, can normally be inlined. Only in case of around advice the `tjp->proceed()` is implemented by an indirect call using a function pointer. Therefore, this operation is currently not inlined, which leads to a noticeably higher resource consumption than for a combined before and after advice. We are currently working on a better code transformation pattern to avoid this problem.

Special Treatment of Dynamic Pointcuts

Currently, the cflow in AspectC++ does not support pointcuts with context variables in the cflow argument. Therefore, a cflow can easily be implemented by a counter that is incremented when the cflow is entered and decremented when the cflow is left. An improved cflow implementation that allows context variables is under development. Here the context will be stored on the call stack.

For the `that()` and `target()` pointcut functions it is sometimes necessary to generate runtime type checks. As we do not require the application code to be compiled with enabled runtime type information, `ac++` generates virtual type condition check functions wherever needed. This operation might change the object layout if the corresponding class had no associated virtual function table before.

Advice Instance Management

Advice code is instantiated at compile time. If the advice depends on the type `JoinPoint`, a join point specific class that implements the required types and functions is generated and used as a template parameter for the advice code.

Deployment and Undeployment

There is no dynamic deployment or undeployment of aspects.