Generic and modular design for FairMQ devices

N. Winckler[1], *M. Al-Turany*[1], *D. Bertini*[1], *R. Karabowicz*[1], *D. Kresan*[1], *A. Lebedev*[1], *A. Manafov*[1], *A. Rybalchenko*[1], *and F. Uhlig*[1]

[1]GSI, Darmstadt, Germany

Introduction

The ability to combine application-specific functionality with independently developed library modules from a variety of sources is a key benefit of long term software projects maintenance and further development. Two main paradigms are available in C++ for achieving such a goal, that is, Object Oriented Programming (OOP) and Generic Programming (GP).

In OOP, libraries usually enforce that the types must be derived from a common abstract base class of the library, providing implementations for a collection of virtual functions. The strength of this paradigm is the dynamic polymorphism where types supplied to a module can vary at runtime. However, module composition is limited since independently produced modules generally do not agree on common abstract interfaces from which supplied types must inherit. On the other hand, the GP paradigm offers mechanisms for producing modules with clean separation, open for extension and without imposing the need to intrusively inherit from a particular abstract base class. However, in the current C++ standard (C++11), GP paradigm only allows static polymorphism, which limits the applications where dynamic polymorphism is required. This limitation will nevertheless disapear with the introduction of C++Concept [1, 2] in the next C++ standard.

Generic FairMQ devices

Policy based class design

In this contribution, we present a policy based design [3] – a design pattern steming from the GP paradigm – for the FairMQ devices [4, 5]. The pattern usually consists of a child class template, called a host class, which inherit from its template parameters, called policy classes. An important aspect is that, the relationship between base and derived class are inverted w.r.t. usual OOP, that is, the base class (host) is the abstract (implicit) interface while the parent classes (policies) supply an implementation set of specific behaviours, which will be inherited by the host class at compile time. The policies are usually split into orthogonal behaviours, and, for a given policy, there can be an unlimited number of implementations. Moreover the host class can be library independent. This design pattern is particularly adapted to the devices of the FairMQ library, since the latter aims to support the process distribution of various experimental group, each depending on different libraries or data types.

Figure 1 shows the class diagram of three generic devices, namely a sampler, a processor and a filesink. Each device inherit from the FairMQDevice abstract class, and from an input policy, an output policy, and eventually a task policy. The FairMQDevice class handle the communication layer. Depending on the device, the input policies can be a file reader, or a deserialization policy and the output policy can be a serialization or a storage policy.

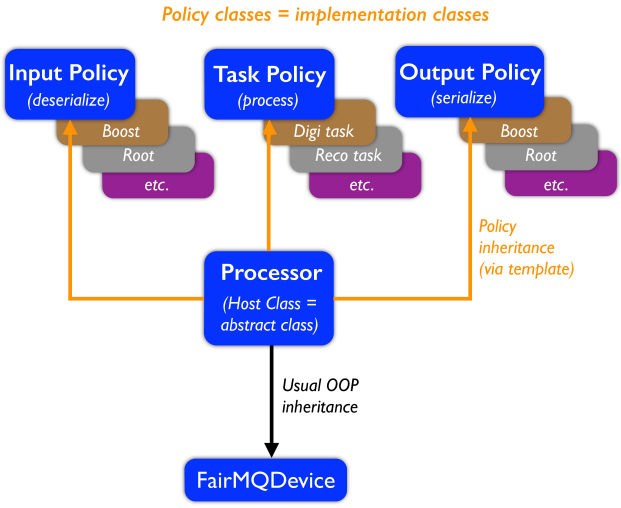

Figure 1: Class diagram of a generic processor.

FairRoot Tutorial 7

An application example of these generic devices can be found in FairRoot/examples/Tutorial7, where different policies are used.

References

[1] B. Stroustrup, et al. “Runtime Concepts for the C++ Standard Template Library”, Proceedings of the 23rd ACM symposium on applied computing (SAC), (2008)

[2] M. Marcus, et al. “Runtime Polymorphic Generic Programming — Mixing Objects and Concepts in ConceptC++”, Multiparadigm Programming 2007: Proceedings of the MPOOL Workshop at ECOOP’07, (2007)

[3] A. Alexandrescu “Modern C++ Design: Generic Programming and Design Patterns Applied” Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)

[4] A. Rybalchenko, and M. Al-Turany “Streaming data processing with FairMQ”, GSI Scientific Report (2013)

DOI:10.15120/GR-2015-1-INFRASTRUCTURE-IT-02

[5] M. Al-Turany et al. “ALFA: A new Framework for ALICE and FAIR experiments”, GSI Scientific Report (2013)

DOI:10.15120/GR-2015-1-INFRASTRUCTURE-IT-02