Most notably:

Implementing a streams design uncovered some drawbacks.

- They provide an adequate model to the primary supervision software needs.
- They can allow for flexible designs, which nevertheless promote coherent maintenance actions in the longer term.
- They can be combined into re-usable blocks, which can easily be built upon, even outside of the initial supervision scope.

In practice, as long as the data produced by a stream is not coupled to a context, the same contract.

mocks, simulated or alternative acquisition sources, or any implementation of the specific stream to be plugged into any arbitrary context, as long as the streams by fully decoupling the definition of a stream and its materialization, it allows any built upon, even outside of the initial supervision scope.

The Streamingpool framework [2, 3] is instrumental to having reusable streams.

The streams paradigm

Streams are ideal to model flows of data, a recurring use-case in the machine controls environment: acquire, transform, deliver. Basic data processing steps, such as conversion, filtering or buffering, are very simply expressed in marble diagrams [1]. Even more complex cases can likewise be visualized and conceived with clarity, such as this analysis helping validate a new BIS hardware loop (SFP), using the operational loop (LHC) as a reference.

Composable architecture

A more functional approach to software development is required to properly work with streams. While there is a significant learning curve to adapt to this model, some benefits are gained from the start:

- Streams organize into a simple hierarchy, representative of the functionality of the system as a whole.
- Each stream is responsible for a single processing step, albeit of arbitrary complexity. This is better achieved with stateless processors, applying pure transformations without state.
- Each stream acts as a clear separation between what is below it and what is above it; each layer does not need to concern itself with any other layer’s business.

Re-usability

The Streamingpool framework [2, 3] is instrumental to having reusable streams. By fully decoupling the definition of a stream and its materialization, it allows any specific stream to be plugged into any arbitrary context, as long as the streams (usually just one) it directly depends on are provided somehow. Those can be mocks, simulated or alternative acquisition sources, or any implementation of the same contract.

In practice, as long as the data produced by a stream is not coupled to a context, this stream can be straightforwardly integrated into another application.

Dealing with legacy

Legitimate circumstances can also bend the architecture and code out of their original shape. While unavoidable, these warts should not leak throughout the code base, neither hinder further maintenance and evolution. The streams hierarchy allows for proper containment of such flaws, both in the architecture and in the code. Besides, legacy paths can be very simply cut out when the circumstances change.

Reference