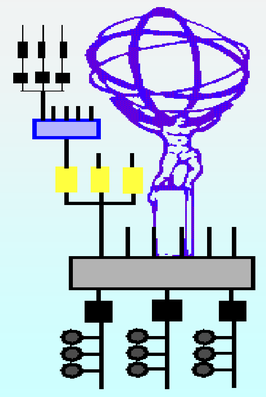
The Error Reporting in the ATLAS TDAQ System

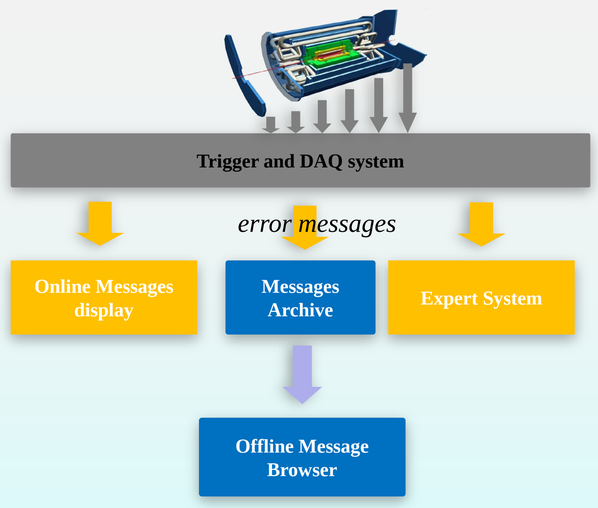


Serguei Kolos, University of California Irvine, USA
Andrei Kazarov, CERN
Lykourgos Papaevgeniou, CERN

1. The ATLAS Error Reporting Architecture

The Atlas experiment
ATLAS is one of the four major experiments at the Large Hadron Collider accelerator at CERN.

The ATLAS TDAQ System
The ATLAS Trigger & Data Acquisition (TDAQ) system transports event data from the 1600 detector read-out links to the mass storage. The system is composed of about 20K applications distributed over 3K computers. Controlling such a system requires a simple, flexible and reliable service for error reporting and handling inside individual application as well as inter-application.



The Error Reporting in the TDAQ system
The Error Reporting System offers an unified system for handling error, warning and debug messages in the TDAQ service. This system is designed to be used both internally by TDAQ applications and to communication between TDAQ applications. Other ATLAS online control and monitoring tools use ERS as one of their main inputs to address system problems in a timely manner and to improve the quality of acquired data.
The actual destination of the error messages depends solely on the run-time environment, in which the online applications are operating. Depending on the actual configuration, the messages which applications send to ERS may end up in a **local file**, a **database**, a **message passing middle-ware**, or in any other output which is supported by ERS. New output devices can be added to ERS as plugins without touching the ERS code.
ERS is available in all programming languages used by the ATLAS software: **C++**, **Java** and **Python**.

2. The Error Reporting Interface

The Streams
Issues are reported to streams. *Stream* is a simple interface with a single function to be used for sending arbitrary problem description to that stream. There are 6 types of streams corresponding to different levels of severity of the reported issues: DEBUG, LOG, IINFO, WARNING, ERROR and FATAL. The severity of any issue is established by the type of stream to which it has been sent

The Configuration
The real destination of the issues sent to a particular stream is defined by the stream configuration. Stream configuration is very flexible and can be defined for each particular stream type at the level of an individual software process by setting the appropriate environment variables. A value of such variable is a comma-separated list of tokens where each token is a well defined ID of a stream implementation class optionally followed by the list of initialization parameters for that implementation.

The Implementations
While ERS provides several default stream implementations, which can be used out of the box, it is an open framework allowing to plug in new implementations at any moment, thus adding new messages destinations without touching the ERS itself. Each *Stream* interface implementation:

- does an action which is specific to that stream. This can be sending the given issue to an appropriate output device, e.g. standard output, database or a mobile phone,
- decides whether the given issue has to be propagated further through the chain of stream implementations or immediately suppressed. This feature allows implementing the issue filtering.

For example the Error stream can be configured via application environment in the following way:

TDAQ_ERS_ERROR="mts, filter(IPC), stderr"

In this case all errors will be sent to the **Message Transfer System (MTS)*** and those which are originated from the Inter Process Communication (IPC) software layer will end up to the **standard error.**

**MTS is the CORBA based distributed message passing middle-ware developed in the ATLAS TDAQ project. It is used for exchanging messages between arbitrary TDAQ applications.*

3. The Issue

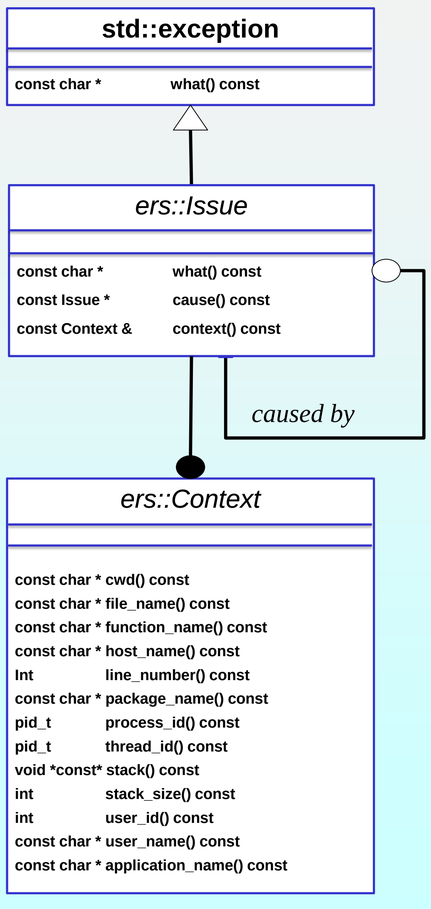
The main class for reporting problems is called Issue.

Inheritance
This class inherits from the std::exception in order to provide compatibility with the standard C++ library. Any custom ERS exception can be caught as std::exception.

Abstraction
The Issue class is abstract. Any software package which has to report an error must declare a specific issue class by inheriting it from the abstract ers::Issue or alternatively reuse an existing issue class from another package. In this case each kind of problem is described by the corresponding C++ class, which facilitates the application of the expert system techniques for errors analysis and decisions taking procedures.

The Context
Each issue has the Context attribute which describes the point where the issue has happened. In C++ this information is provided by the special macro ERS_HERE, which must be the first argument to each Issue constructor. In Java and Python such information is extracted in the Issue constructor itself, so no additional arguments is used.

The Chains of Issues
An issue may contain a pointer to another issue which provides more detailed information on the cause of the problem. Such chain of issue may have arbitrary depth thus giving any required level of detailed for the problem description.



4. Declaring Issues

Having strongly typed errors is highly desirable for simplifying the system maintainability and absolutely indispensable for replacing human operator with expert system. On the other hand writing issues classes declarations would have been tedious and error prone.

MACRO to the rescue
ERS uses the mind breaking **BOOST Preprocessor** package. Despite conventional opinion the usage of macro constructs in C++ may be extremely useful and convenient, drastically reducing amount of code which has to be written and improving the code quality and maintainability.

```
ERS_DECLARE_ISSUE(
      namespace, class_name, message, class_attributes)

ERS_DECLARE_ISSUE_BASE(
      namespace, class_name, base_class_name,
      message, class_attributes, base_class_attributes)
```

Examples

```
ERS_DECLARE_ISSUE(
      io,
      FileIssue,
      "Basic issue with '" << file_name << "' file",
      ((const char *)file_name ))
```

```
ERS_DECLARE_ISSUE_BASE(
      io,
      PermissionDenied,
      FileIssue,
      "Insufficient privileges for " << opeartion << " '" << file_name
            << "' file which has " << permissions << " permissions",
      ((const char *)file_name ),
      ((int)permissions) ((const char*)operation))
```

```
ERS_DECLARE_ISSUE_BASE(
      io,
      CantOpenFile,
      FileIssue,
      "Can't open file " '" << file_name << "' file",
      ((const char *)file_name ),
      ERS_EMPTY )
```

5. Using ERS for error catching and reporting

ERS provides two ways for reporting messages to different streams:

- There are 6 functions for reporting issues to different ERS streams, e.g. ***ers::debug(), ers::log(), ers::info(), ers::warning(), ers::error()*** and ***ers::fatal()***

```
try {
    open_file( file_name );
}
catch ( io::PermissionDenied & ex ) {
    ers::warning(io::CantOpenFile( ERS_HERE, file_name, ex ));
}
catch ( io::FileIssue & ex ) {
    ers::error( ex );
}
catch ( std::exception & ex ) {
    ers::fatal(io::FileIssue( ERS_HERE, file_name, ex ));
}
```

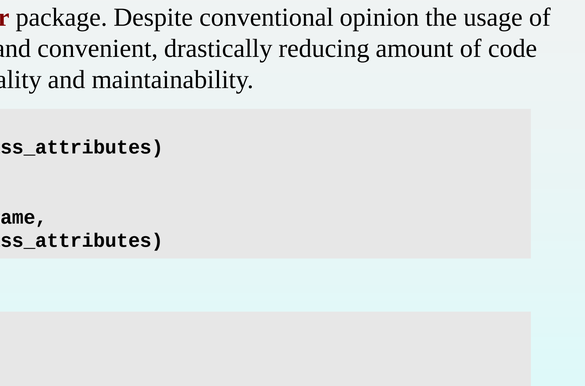
- Alternatively, any "harmless" events can be reported via macro provided by ERS, i.e. ERS_DEBUG, ERS_LOG, ERS_INFO. These macro can be used to send arbitrary information without defining any new issue type.

```
ERS_DEBUG( 1, "test debug output " << 123 << " which shows how to use debug macro" );

ERS_LOG( "So far " << event_number << " events have been collected");

ERS_INFO( "The run " << run_number << " has been started" );
```

This separation is very important for enforcing strong typing for the issues which report essential events to the "harmful" streams, i.e. Warning, Error and Fatal. This allows reliable usage of such issues in the reasoning systems, the TDAQ is using for the system control and monitoring.

6. Error Reporting in the Past and Future Runs

Run 1: 2010 - 2013
For the Run 1 about 30M ERS messages have been produced by the TDAQ system. All those messages have been archived to the Oracle database. The dedicated Log Browser application has been provided for browsing the content of that database.
At run time about 10 different types of ERS issues have been used by the ATLAS online Expert System for error detection and handling during data taking.



Run 2: 2015-2018
For the upcoming data taking period the archived system back-end will be changed to Splunk (http://www.splunk.com), which is an easy-to-use web interface and powerful enterprise platform for analyzing machine data. This approach has an obvious advantage of having access to the archived messages from all over the world without the need of installing any ATLAS specific software.