A Software Framework for KM3NeT

Claudio Kopper [1]

Erlangen Centre for Astroparticle Physics (ECAP), Univ. Erlangen-Nürnberg, Erwin-Rommel-Str. 1, 91058 Erlangen, Germany

Abstract

Modular software frameworks have become indispensable for large-scale experiments like the KM3NeT neutrino telescope. This article gives a generic definition of a software framework and presents an adaptation of *IceTray*, the framework currently in use by the IceCube collaboration, for water-based detectors.

Key words: KM3NeT, software framework, IceTray, IceCube, analysis, software
PACS: 95.55.Vj, 29.85.-c, 29.85.Fj

1. Introduction

A major physics experiment like the KM3NeT neutrino telescope [1] requires an immense amount of hardware development effort. However, to make use of the telescope, software that allows for running the experiment and, most importantly, to transform the raw data to high-level physics information is needed. Such software cannot be written by a single person and thus has to be created in a collaborative effort.

A software developer should not need to know every single line of code of a particular program to be able to extend it. Most of the time, the task is to implement or change a single algorithm only, without modifying the rest of the reconstruction software.

It is thus necessary to write modular code with a predefined data flow from the very beginning. Such program modularisation and the definition of a corresponding data flow are the essential elements provided by a *software framework*.

Email address:
claudio.kopper@physik.uni-erlangen.de; Tel: +49 9131 85 27 150; Fax: +49 9131 85 28 774 (Claudio Kopper).
[1] for the KM3NeT Consortium

This article first gives a definition of a software framework and illustrates why frameworks have become indispensable for major physics experiments. Finally, an adaptation of the *IceTray* [2] framework to KM3NeT is presented. *IceTray* is the framework currently in use by the IceCube collaboration [3], who granted KM3NeT access to the IceTray code for assessment.

2. Definition of a framework

A possible definition of a software framework is a set of rules, interfaces and services provided to the programmer, who can use it to perform a set of tasks.

In a modular framework, the user should only have to change a few lines in a steering file to modify an analysis chain. This is illustrated in Fig. 1: An existing analysis is changed by adding a further module.

Frameworks should not be confused with class libraries. A class library, like a framework, provides a set of interfaces and services. It does not, however, enforce a particular program logic and leaves it to the individual programmer's code to specify the logic. In a framework, the program logic is pre-

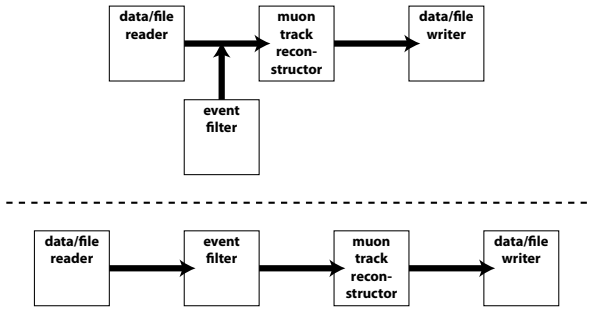

Fig. 1. Simplified reconstruction chains consisting of *modules*, with data flowing from one module to the next one. The first chain consists of three modules. In a framework, tasks like adding modules can easily be performed. In this example, an event filter is inserted at a particular point in the first chain. The resulting reconstruction chain is shown below the dashed line.

defined, which makes it easy to extend. A programmer implements algorithms, whereas the framework specifies how these algorithms interact.

Typically, frameworks enforce this overall design. At the same time, they avoid being too restrictive, to allow for the necessary freedom to the developers and end-users. [2] This is achieved by using a modular approach, where each subsystem can be modified, added or replaced without altering the others.

Similar to a computer operating system, a framework only provides a high-level interface to make life easier for programmers and end-users. To be able to do analysis, the physics contents have to be added to the framework in the form of algorithms. To complete the analogy, a computer is useless if it is just installed with a bare system kernel. It can only be used after installing applications like a web browser or an office suite. These applications then make use of the hardware by accessing the abstraction layer provided by the system kernel. Of course, in principle, it is possible to use a computer without an operating system by creating an application that accesses all hardware directly. But clearly, such an approach would neither be modular nor easy to use or develop.

Using a framework saves the physicist a lot of time in learning how to use a particular software package every time s/he wants to perform a new task on a set of data. All algorithms are implemented in modules having the exact same structure. This makes the code easier to read and understand. The software user can thus start to work on physics problems with a significantly shorter learning period.

A modular structure also leads to more flexibility. Systematic analyses can be performed more easily, as the framework explicitly allows the user to change the program flow.

Frameworks facilitate the collaboration between different groups doing software development. As all modules have the same format, they can be easily exchanged. This is especially advantageous when doing cross-checks and also helps implementing an overall quality control scheme: Code that is broken down into small independent units can be more easily reviewed by others than a single monolithic program.

3. IceTray

IceTray, the software framework developed and used by the IceCube collaboration [3], provides all the features mentioned in the previous section. Developers create *modules*, which can be dynamically linked into the framework. They contain the actual algorithms responsible for event reconstruction or simulation. Data is passed from module to module in data containers called *frames*. Each frame describes a single detector event. A frame consists of a list of name-object tuples. Except for an I/O method, there are no requirements imposed on the objects stored in a frame. This design decision makes the framework easily extendable.

The framework already provides modules for reading in and writing out data. These modules can be used at any time in the reconstruction chain, so that data can be written out at any stage. Though IceTray uses its own data format based on routines from the *boost.org* libraries [4], it is possible to write reader modules for any other data format. This has been demonstrated by providing a reader module for standard ANTARES ROOT-files as produced by the current ANTARES online DAQ system [5]. It should be noted that, in principle, it is never necessary to write out any intermediate data, as data input and output is done in dedicated modules. A full Monte Carlo simulation of events and their reconstruction can thus be done in IceTray without ever writing temporary files to disk. Of course it

[2] In an object oriented language like C++, a particular design can be enforced by using the concept of *encapsulation*: the inner workings of a class should always be hidden from other classes. Another important concept, called *polymorphism*, allows the framework to be extended in a well-defined way: only a small set of abstract classes is exposed from the framework. These classes are then to be overwritten when implementing an algorithm.

is still possible to produce intermediate data files. Data can be written out at any point in the analysis chain at the user's discretion.

The order of modules can be defined when initialising the framework. This is typically done using the *Python* scripting language [6]. In addition to Python, compiled C++ code can be used when writing IceTray programs.

IceTray comes with *dataclasses*, containing class definitions usable for storing positions, directions, particle tracks, OM properties and all other data necessary during event reconstruction and simulation. These classes are usable for KM3NeT with only a few minor modifications. These modifications are mostly necessary because the developers of IceTray assumed that all optical modules would look either straight downwards or upwards. This is not necessarily true for water-based neutrino telescopes, where OMs can face in arbitrary directions.

A few simple non-physics modules, such as a generic event selector template and a ROOT tree writer can also be re-used for KM3NeT. None of these helper modules contain any physics relevant code.

IceTray includes an installation system for almost all external packages it depends on. These dependencies, like ROOT and several other libraries, are installed prior to compiling IceTray. The installation is done using a variant of the *ports* installation system [7]. It substantially simplifies this process, as all packages can be downloaded and compiled with only two shell commands.

4. Adaptation of IceTray for ANTARES and KM3NeT

To enable IceTray to work with the data that is currently available, a considerable amount of ANTARES code was re-written or encapsulated into modules. Most of these modules can also be used for KM3NeT without major changes.

In addition to the DAQ file reader module mentioned above, a module that permits reading the standard ANTARES Monte Carlo file format was developed. It enables the user to read data produced by the standard ANTARES Monte Carlo packages [8]. Thus, all of the currently existing Monte Carlo data can be used with IceTray.

To be able to compare Monte Carlo events to real data, full simulations of the Optical Module (OM), the digitisation electronics [9] and the environmental background noise at the detector site [10] are needed. All these aspects of the simulation chain have been encapsulated into modules: The OM simulator uses a parameterised approach to simulate the transit time spread and the amplitude smearing of the photomultiplier tubes (PMTs). As there are slightly different parameterisations in use in different ANTARES tools, the user can choose between different options. The simulation of the PMT signal digitization is fully compatible with the one currently in use by ANTARES. Environmental background hits (from K40 decays and bioluminescence) are generated with a fixed user-configurable rate per OM. Alternatively, the noise rate per OM can be taken from a real detector run to approximate the real noise distribution within the detector volume.

When using Monte Carlo data, it is necessary to be able to simulate different detector trigger conditions. For these purposes, the complete ANTARES core trigger code was encapsulated into an IceTray module. This module receives the current event data and converts it into the format used internally by the ANTARES trigger. The original trigger code can thus be used unchanged, which allows for easy maintenance and updates in case new trigger algorithms are added to the ANTARES DAQ. After calling the trigger routines, the module converts all results back to a data format compatible with IceTray and passes all information to the next module.

Reconstruction of events is typically done using a simple pre-fit algorithm in conjunction with a more sophisticated reconstruction algorithm, possibly combined with an iterative series of hit selections. Therefore, such reconstruction strategies can be broken down into a number of modules. This has been done for the standard ANTARES muon reconstruction strategy [11]: All pre-fit, intermediate fit and hit selection steps have been re-written as IceTray modules. Track candidates are passed from each fit module to the next one, where they can be used as starting points. The resulting track candidate is then passed to a final fit algorithm which was also converted into an IceTray module.

During the implementation of this reconstruction strategy, a module named *SubTray* was implemented which allows for a number of modules in a *Python* script to be used as a single module in another script. With this functionality, experts can create whole analysis or simulation tasks and encapsulate them into prefabricated scripts. These can then be used as single modules by others without them knowing every single detail. This is useful for

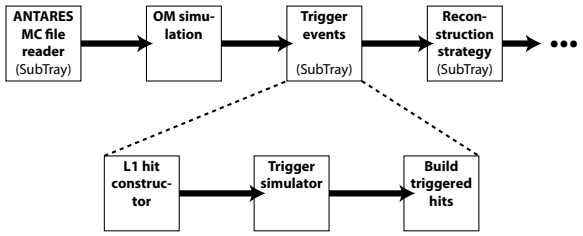

Fig. 2. An example module chain for ANTARES Monte Carlo data which uses *SubTrays*. Most tasks are performed by dedicated SubTray modules. For the *Trigger events* module, the actual SubTray module chain is shown. In this chain, a list of coincident hits on a floor (*L1 hits*) is built. These hits are then used by the trigger module, which in turn produces a list of *triggered hits*. As this list only contains one hit per coincidence, the full list of triggered hits is built as a last step. This illustrates that by using a SubTray, more complicated tasks consisting of multiple modules can be combined into single re-usable scripts.

reconstruction strategies, as users just have to add a single module to their analysis chain instead of every single fitting and hit selection step. There are further applications for this module in other parts of the framework, e.g. when combining multiple simulation steps into a single event builder module. Figure 2 shows an example of a module chain using *SubTray* modules. It should be noted that the module does not limit experts from taking full control over every aspect of their simulation or reconstruction task: The module chain from a *SubTray* script can always be inserted directly into the main script.

5. Summary

Software frameworks are essential parts of every major physics experiment. KM3NeT is considering the adoption of a framework in the current phase of the design study [12]. If Monte Carlo studies done now are performed in the same environment as the real reconstruction later on, the results are comparable more easily.

Another important point is the steeper learning curve for the developer in learning how to use a particular software. As the program structure in a framework always stays the same, the programmer has to learn it only once in the beginning.

Using IceTray in particular has the advantage that this framework is already in use for neutrino telescopes, is actively developed and has been thoroughly tested. Using the same data-format as IceCube also opens future possibilities for sharing data.

A considerable amount of work has been invested to make IceTray compatible with ANTARES data and to modularise the most commonly used algorithms.

6. Acknowledgments

The author wishes to thank the IceCube collaboration for their co-operation, especially for providing access to the IceTray framework source code. This work has been funded by the EU in the framework of the KM3NeT Design Study, FP6 contract 011937.

References

[1] KM3NeT Consortium, P. Bagley et al., *KM3NeT Conceptual Design for a Deep-Sea Research Infrastructure Incorporating a Very Large Volume Neutrino Telescope in the Mediterranean Sea*, 2008, available from `http://www.km3net.org/CDR/CDR-KM3NeT.pdf`

[2] IceCube Coll., T. DeYoung, *IceTray: A Software Framework for IceCube*, International Conference on Computing in High-Energy Physics and Nuclear Physics (CHEP2004), 2004, available from `http://www.chep2004.org/`

[3] IceCube Coll., J. Ahrens et al., *Preliminary Design Report*, 2001, available from `http://www.icecube.wisc.edu/science/publications/pdd/`

[4] available from `http://www.boost.org/`

[5] ANTARES Coll., J. A. Aguilar et al., *The data acquisition system for the ANTARES neutrino telescope*, Nucl. Instrum. Meth. A **570** (2007) 107 [arXiv:astro-ph/0610029]

[6] see `http://www.python.org/`

[7] see `http://www.macports.org/`

[8] D. Bailey, *Monte Carlo tools and analysis methods for understanding the ANTARES experiment and predicting its sensitivity to Dark Matter*, PhD thesis, Wolfson College, Oxford, 2002

[9] ANTARES Coll., *Technical Design Report of the ANTARES 0.1 km2 project*, 2001, available from `http://antares.in2p3.fr/Publications/`

[10] ANTARES Coll., P. Amram et al., *Background light in potential sites for the ANTARES undersea neutrino telescope*, Astropart. Phys. **13** (2000) 127 [arXiv:astro-ph/9910170]

[11] A. Heijboer, *Track Reconstruction and Point Source Searches with ANTARES*, PhD thesis, Universiteit van Amsterdam, 2004

[12] see `http://www.km3net.org/`