A dimension aware evaluator for High Energy Physics applications

M. Clemencic and B. Couturier

May 8, 2023

CERN - LHCb

Table of contents

Why?

Who needs an evaluator?

Evaluator

A utility that takes a string representing a mathematical expression and returns the number obtained by evaluating the expression.

- Useful when one needs to convert expressions to numbers at run time
  - e.g. from user inputs in the form of complex configuration files
- In LHCb it is usually needed in the description of the detector geometry
  - the geometry description is in XML files
  - basic quantities are defined as constants
  - other quantities are derived using expressions
  - expressions are evaluated at runtime after parsing the XML files

What is the problem with dimensions?

Physical Quantity

A physical quantity is identified by a magnitude and a unit of measurement that tells which type of quantity we are referring to (length, mass, time, ...) and how the magnitude value compares with other quantities of the same type.

Mathematical operations between physical quantities follow precise rules

- additions and subtractions apply only to quantities of the same type
- multiplications and divisions change the type of quantities
  - $\text{length} * \text{length} \rightarrow \text{area}$
  - $\text{length}/\text{time} \rightarrow \text{speed}$
- numerical values have to be scaled according to their units of measurement
  - $1\ \text{m} + 1\ \text{mm} \rightarrow 1.001\ \text{m} \equiv 1001\ \text{mm}$

How it has been addressed so far?

Physical quantities operation rules where usually addressed by

- choosing reference units of measurement
  - e.g. all lengths have to be expressed in mm
- helper variables used to convert to/from the reference units
  - `1 * m + 1 * mm = 1001 * mm = 1001`
  - `length_in_m = my_length / m`
- hope that nobody makes mistakes
  - nothing prevents the invalid operation `1 * m + 1 * s`

This is, of course, error prone, as the Mars Climate Orbiter crash demonstrated.

LHCb specific problem

- Detector description migrated from custom library to DD4hep
  - change of convention for reference units
- Units helpers are not enough
  - `1 * mm` and `1` are indistinguishable
- Some common mistakes
  - missing units in expressions
  - wrong scaling factors crossing libraries boundaries
- Work around
  - compile DD4hep with LHCb reference units convention

The tools

Libraries

- Libraries for different languages
  - C++: Boost.units, standardization proposal P1935, ...
  - Python: Pint, quantities, ...
  - Lua: lua-physical
  - Rust: uom
- Different pros and cons
- An evaluator is between compiled and interpreted
  - the interface needs compile time checks
  - the expression is evaluated at run time

Design

What and how

- C++
  - validation and scaling needed at compile time
- Run time evaluation
  - reject invalid operations
  - correctly handles scaling between units
- Bridge the gap between compile time and run time
  - convert requested C++ unit to evaluator expression
  - ask the evaluator to validate and convert to number
  - C++ takes the number and adds the compile time unit

Implementation

Options

- Evaluator implementation
  - modify an existing evaluator (e.g. CLHEP Evaluator)
  - implement a new evaluator
  - leverage on an embedded language
- C++ units library
  - it must have a conversion to string of the unit

Choices

- Evaluator implementation: Lua + lua-physical
  - Lua is a language with a small runtime designed to be embeddable
  - not many physical quantities libraries and lua-physical is OK
- C++ units library: Boost.units
  - mp-units (from P1935) is more interesting but requires C++20
  - Boost.units is mature and supports conversion to string

Technical obstacles

- Deployment of Lua modules
  - embed lua-physical code in the compiled library
- Naming conventions: lua-physical prefixes units with "_"
  - patched to be compatible with LHCb ($m \leftrightarrow$ `_m`)
- Conversion between C++ units and evaluator expression
  - Boost.unit conversion to string is for humans
    - e.g. `m kg s^-2` instead of `m * kg * s^-2`
  - it's possible to implement a dedicated conversion
    - for a proof of concept we *massaged* the string

Using the library

(from unit tests)

```
TEST_CASE("examples") {
  LuaEvaluator l;

  CHECK(l.eval<km>("1 * au").value() == 149597870.700);
  CHECK(l.eval<km / h>("1 * km/h").value() == 1.);
  CHECK(l.eval<metre / second>("1 * mi/h").value()
        == Approx(0.44704));

  CHECK_THROWS(l.eval<metre>("10 * h"));
}
```

The interface

```
class LuaEvaluator {
public:
  double eval(std::string_view expression,
              std::string_view result_unit);

  template <auto unit>  // requires C++17
  auto eval(const std::string_view expression) {
    const auto value = eval(expression,
                            details::unit_repr<unit>());
    return value * unit;
  }
};
```

constructor, destructor and data members are omitted

Summary

Summary

- We produced a proof of concept implementation
  - combining Boost.units, Lua and lua-physical
  - we can evaluate almost all expressions in LHCb detector description XML
- Some limitations
  - some useful operations on units are not available (e.g. $sqrt$)
  - conversion from Boost.unit string to Lua is fragile and incomplete
- Next steps
  - estimate the cost of integrating the new evaluator in the framework