

A Survey of Physical Unit Handling Techniques in Ada

Christoph Grein¹, Dmitry A. Kazakov², and Fraser Wilson³

¹ ESG-Elektroniksystem- und Logistik-GmbH, D-81675 München

² cbb software GmbH, D-23560 Lübeck

³ Anago b.v., NL-3995AA Houten

Abstract. There has always been a demand to be able to compute with physical items where dimensional correctness is checked. A survey of methods is presented how to do this with the features of Ada. Compile-time methods use the type checking mechanism whereas run-time methods use additional components to represent dimensions.

1 Introduction

When dealing with physical equations, physicists are used to checking their results for dimensional correctness. So they feel a certain loss when they transfer their equations to computers since none of the commonly used programming languages provide features to deal with physical dimensions. Programmers resort instead to work-arounds within the language-provided features to keep track of dimensions or do completely without them.

In the following chapters, a few methods are presented how the features of Ada can be used to compute with dimensions.

We would however like to state that it is not the wrong equations that introduce the actual problems in big software systems (these can be spotted by code inspections); their causes normally are buried much earlier in wrong system or software designs, as has been shown by some recent spectacular failures.

Mars Lander's problem was the mixing of metric and British units in the communication between two CSCIs, which none of these methods would have been able to avoid since they apply only within a CSCI (Computer Software Configuration Item – a separate program communicating with other programs via e.g. global memory).

The ideas to import dimensions into Ada (or other programming languages) are not new. Do-While Jones [1, 2] published such a method using private types, which however becomes unwieldy when dimensions are mixed; it also does not include mathematical functions.

Gehani [3] compared the appearance of expressions in Ada with a hypothetical language which includes dimensions as data attributes and came to the conclusion that only the latter method can solve all problems elegantly and satisfactorily. Since at that time, he considered an Ada language change as most improbable (the Ada9X process has proven him correct), he proposed a solution how correctness of physical expressions can be checked during run-time (by using discriminants).

Gehani's method suffers however from its notation of declarations and statements that differs considerably from the common technical and scientific one. The nota-

tional problems of both Gehani's and Jones' methods can be cured if Ada's features are used cleverly as Hilfinger [4] has shown, who based his ideas on Gehani's work.

In passing, we mention as a third method beside the two above: the use of pre-processors. Schneider [5] presents such a pre-processor for Pascal, implemented in C. He also provides an overview over attempts to include dimensions into programming languages. For further references, see his paper.

A universal unit support faces contradictory requirements which are difficult if not impossible to satisfy:

1. **Compile-Time Checks.** The dimension errors should be discovered as early as possible. For example if dimensions are known at compile time, then all unit errors shall be detected at this stage.
2. **No Memory / Speed Overhead at Run-Time.** If dimension information is not used at run-time (because of 1), then at run-time neither memory should be allocated to keep it, nor checks should be made.
3. **Support for Derived Types.** Dimensioned operations should involve not only the isles of scalar types. There should be a way to declare other dimensioned types, for example, an array of dimensioned scalars; a dimensioned record type; a dimensioned private type, with all necessary cross operations.
4. **Generic Programming.** Here generic is used in a wider sense, as an ability to write code dealing with items of types from some type sets. In our case it means items of different dimension. For instance, it should be possible to write a dimension-aware integration program, which would work for all valid combinations of dimensions.
5. **No Precision / Range Loss.** Very often dimensioned values are measured in units different from the base ones. There should be a way to deal with such values without a necessity to convert them to the base units, which is inevitably connected with a precision / range loss. It is hard to expect and wrong to require that an astrophysical program would use meters to calculate distances between galaxies.
6. **Scales.** A more generalized variant of 5 is when data are measured in scales different from standard, or additional constraints are imposed by the nature of the measured values. Examples of scales are logarithmic values, Celsius temperature scale, time versus duration etc. Though values of different scales might have the same units, they cannot be mixed in standard arithmetic operations. For instance in case of time and duration, *time* – *time* gives *duration*, *time* + *time* is not defined.

All varieties of possible solutions of the dimension problem can be roughly subdivided into two big classes according to the answer to the question: "Should the dimension information be mapped to different types?"

A positive answer would automatically provide requirements 1 and 2 with little or no efforts, because types are fully checkable at compile-time in Ada. Generics can be used to provide requirement 4 by transferring the dimension as a generic parameter.

In case of a negative answer other questions arise: "Can the method still allow requirements 1 and 2? Should the dimension information be a part of the type?" When it is not a part of the type, then we loose requirement 2 without any hope. In other words: To fully support requirement 4, one should probably have the dimension information as a type parameter.

The conflict between the requirements 1 and 2 on one side and requirement 4 on the other is the source of the problem of dimension handling.