Computer Standards & Interfaces 56 (2018) 144–147

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Architecture for software-assisted quantity calculus☆

David Flater

National Institute of Standards and Technology, 100 Bureau Dr, Gaithersburg, MD 20899, USA

ARTICLE INFO

Keywords:
SI
Quantity
Unit
Uncertainty
Value
Unit 1

ABSTRACT

A quantity value, such as 5 kg, consists of a number and a reference (often an International System of Units (SI) unit) that together express the magnitude of a quantity. Many software libraries, packages, and ontologies that implement "quantities and units" functions are available. Although all of them begin with SI and associated practices, they differ in how they address issues such as *ad hoc* counting units, ratios of two quantities of the same kind, and uncertainty. This short article describes an architecture that addresses the complete set of functions in a simple and consistent fashion. Its goal is to encourage more convergent thinking about the functions and the underlying concepts so that the many disparate implementations, present and future, will become more consistent with one another.

Published by Elsevier B.V.

1. Introduction

In the scientific conventions of the early 19th century, mathematical operations applied only to numbers, and units of measure were just information that described what the numbers represented. This belief gave way to the practice of including units of measure within the scope of the mathematical operations, thereby formalizing the method of working with combinations of units. The resulting *quantity calculus* methodically determines the units of derived quantities and protects us from the error of computing nonsensical combinations of quantities that have different dimensions [2]. For example, if a property line is moved by 3 m, we are allowed to add 3 m to the width of the lot, but we cannot add 3 m to the lot size that is measured in m^2. We must multiply 3 m by the depth of the lot (x m) to obtain the change to the lot size ($3x$ m^2).

Many software libraries and packages that implement "quantities and units" (Q&U) functions are available; for example:[1]

- Mathematica, quantities and units in the Wolfram language [3];
- Maxima package ezunits [4];
- Modelica, physical variables and SIunits library [5];
- LabVIEW unit labels [6];
- MathCad units [7];
- GNU Units [8];
- UDUNITS [9];
- R packages for units of measure (e.g., units and udunits2 [10]);
- Ruby gems for units of measure (e.g., ruby-units [11], phys-units [12], and unitwise [13]; there are at least 10);
- C++ libraries for units of measure (e.g., Boost.Units [14]);
- F# units of measure (a built-in language feature) [15];
- Python package numericalunits [16]; and
- Julia packages SIUnit [17] and Unitful [18].

A comparable number of formal models and ontologies related to quantities and units exist; for example:

- Conceptual model of the International Vocabulary of Metrology (VIM) [19, Annex A];
- Dybkaer's Ontology on Property [20];
- Unified Code for Units of Measure (UCUM) [21];
- Quantities and Units of Measure Ontology Standard (QUOMOS) [22];
- Units Markup Language (UnitsML) [23];
- Quantities, Units, Dimensions, and Data Types Ontologies (QUDT) [24];
- Quantities, Units, Dimensions, Values (QUDV) in Systems Modeling Language (SysML) [25]; and
- Ontology of units of Measure and related concepts (OM) [26].

☆ This article is an abbreviated and revised extract from a National Institute of Standards and Technology (NIST) technical report with the same title [1]. Official contributions of NIST are not subject to copyright in the United States.

E-mail address: david.flater@nist.gov

[1] Certain commercial entities or software are identified in order to describe a concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities or software are necessarily the best available for the purpose.

https://doi.org/10.1016/j.csi.2017.10.002
Received 9 August 2017; Received in revised form 27 September 2017; Accepted 10 October 2017
Available online 13 October 2017
0920-5489/Published by Elsevier B.V.

Although all of the above libraries, packages, models, and ontologies begin with the International System of Units (SI) [27] and associated practices, they differ in how they address issues such as *ad hoc* counting units, ratios of two quantities of the same kind, and uncertainty. These issues and others that have undermined the functionality and consistency of software for metrology have been discussed in related work [7,28].

This article describes an architecture that addresses the complete set of functions for Q&U software in a simple and consistent fashion. The goal is not to produce *yet another* implementation or framework to compete in an already overcrowded arena, but to identify the major architectural features that have proven to be important or sorely needed in this author's experience, and thereby encourage more convergent thinking and improved compatibility among different implementations in the future.

The building blocks of the architecture are:

- A catalog of recognized units and numerical prefixes;
- Values (in the measurement sense), including derived values and compound values;
- Probability distributions to characterize uncertain values, with automated propagation of distributions to derived values; and
- An extensible type system for specializations of the SI unit 1.

Section 2 through Section 5 address each of these in turn. Additional recommendations concerning numeric data types are given in Section 6. Section 7 summarizes.

2. Units

Quite simply, one cannot begin to implement quantity calculus without first having a catalog of units that the software will recognize and refer to. We need not belabor the point, but all Q&U software must at least recognize the standard base units, derived units, and numerical prefixes (k, M, etc.) that are identified in the SI brochure [27] and be capable of converting "non-coherent" units to their standard equivalents.

3. Values

The following terms are defined by the 3rd edition of the VIM [19].

quantity: property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference.

[The "reference" is typically an expression in terms of SI units.]

quantity value: number and reference together expressing magnitude of a **quantity**.
measurement result: set of **quantity values** being attributed to a **measurand** together with any other available relevant information.

[The "set of quantity values" is intended to accommodate uncertainty, given that a single true quantity value generally cannot be determined.]

A *value* in a software implementation of quantity calculus represents one or more measurement results with one or more **quantity value** variables. Those with only one such variable (the usual case) are *simple values*; the rest are *compound values*.

A *base value* represents immediate results of a measurement. A *derived value* is the result (output) of a function of other values (the inputs).

Figure 1 illustrates the concepts. On the left side, two base values, a speed expressed in meters per second and a duration expressed in seconds, are input to a function that multiplies them together, producing a derived value that is expressed in meters. On the right side, polar coordinates are given as an example of a compound value, because in most applications it is necessary to keep the radial coordinate and the angular coordinate together.

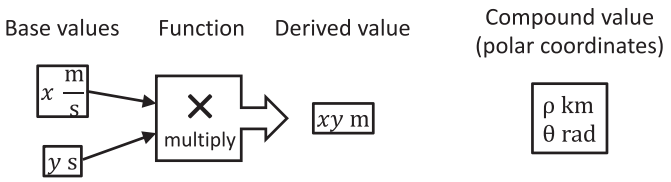

Fig. 1. *Values* in a software implementation of quantity calculus.

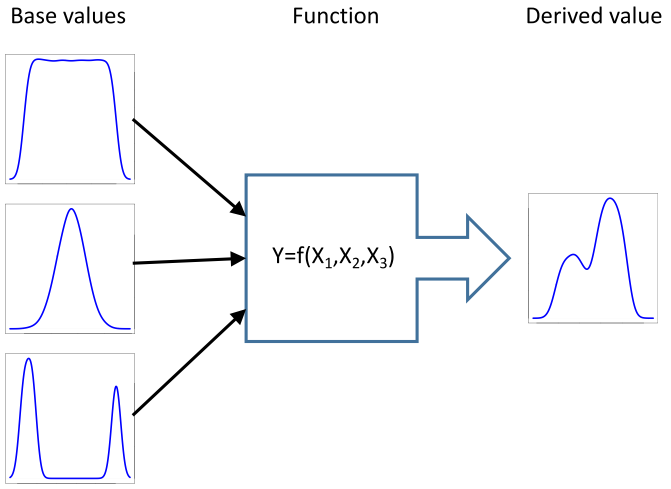

Fig. 2. Propagation of the distributions of base values through a function to the distribution of a derived value.

Like the catalog of SI units, simple values and value functions are essential to all software implementations of quantity calculus. Compound values, on the other hand, are not universally supported.

In the architecture described by this article, values are represented by distributions, as explained in the next section.

4. Characterizing and propagating uncertainty

Uncertainty too often is completely ignored or is handled in a limited way in available Q&U software.

Expressing uncertainty using standard deviations and multiples thereof involves simplifying assumptions that are unnecessary when quantity calculus is automated. The more general approach is to propagate probability distributions (see Figure 2) [29,30]. Probability distributions replace both a point estimate and its standard uncertainty. They may be continuous (for dimensional quantities and ratios) or discrete (for counted quantities). They may be univariate (for simple values) or multivariate (for compound values). They may be unimodal (for simple estimates) or multimodal (for when a single estimate would be misleading). And if a user wishes to have a quantity value with zero uncertainty, such as a trivial count or a defined constant, it can be assigned a degenerate (deterministic) distribution that has only one possible value.

Propagation of distributions can be implemented in two distinct ways:

1. A distribution is a black box software function that exposes only the interface to get sampled quantity values. A quantity value is characterized by a histogram or kernel density plot derived from a sufficiently large sample, and an interval is produced using statistical methods. When a derived value is sampled, its value is calculated using samples from other values as inputs to its function.
2. A distribution is a symbolic mathematical function that is either represented directly by the software or suitably approximated (e.g., with Chebfun [31]). Sampling is not required; instead, a quantity value is plotted directly, and intervals and derived values are derived by operating algebraically on the functions themselves.

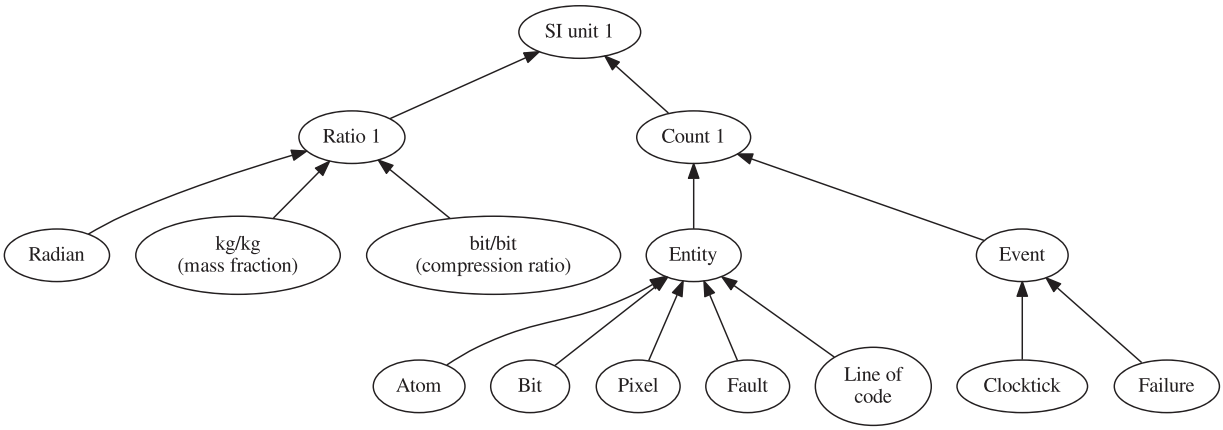

Fig. 3. Example types of non-dimensional units.

A given system could implement both approaches. For example, the black box approach could be used as a fallback when it becomes infeasible to represent or derive a distribution algebraically.

Existing implementations in software include Uncertain⟨*T*⟩ [32], the NIST Uncertainty Machine [33], and many others [34]. Unfortunately, implementations are seldom pre-integrated with Q&U software.

5. Subtyping unit 1

A quantity in SI can be stated as the product of a numerical value and a unit of measurement. The unit is derived from the seven SI base quantities, which measure seven different physical dimensions. However, many kinds of quantities have no extent in any of those dimensions. For example, a *counted quantity* is a number of some distinguishable kind of thing, such as 32 bits. Ratios of two quantities of the same kind, such as mass fractions (kg/kg), are in another major category of dimensionless quantities that includes all of what are variously referred to as 'characteristic numbers,' 'similarity criteria,' and 'composed dimensionless quantities' [35].

The SI unit for dimensionless quantities is the special unit 1, which either is derived from base quantities by raising them to the power 0 or is an implicit, zeroth base quantity [19,27]. Regardless of its derivation, when this same unit is used for all dimensionless quantities, the formalisms of quantity calculus will not detect or prevent mistakes such as converting an amount of data (measured in bits) into an angle (measured in radians), as might occur if it were erroneously supplied as the input to a trigonometric function. Such mistakes might go undetected if they were embedded within a more complex computation. The risk of confusing different kinds of quantities that happen to be expressed in the same units is not limited to dimensionless quantities, but it is worse because of the sheer number of different kinds of quantities that all must refer to the unit 1.

Unfortunately for computer science, "amount of data" is not an SI dimension, and bits and bytes are not SI units. To avoid user surprise at the canonical treatment of amounts of data and other dimensionless quantities in SI, software libraries and packages that implement Q&U functions often apply workarounds. Different software has applied different workarounds, creating subtle problems for transfer of scientific data.

Here, again, we seek to encourage more convergent thinking by introducing a sufficiently general model. It is a formalization, extension, and modification of an approach that was initiated by Mohr and Phillips [36]. The idea is to extend quantity calculus to support subtyping of the special unit 1. This enables traceability to SI to occur not only through direct reference to SI units (the identity relationship), but also through subtyping (the generalization/specialization relationship).

The top levels of a type system that interprets dimensionless units are shown in Figure 3. Following Ref. [36], countable things are subtyped into entities and events. Although certain terms in natural language refer simultaneously to entities and events, it is important in a measurement context to distinguish which is being counted. For example, the number of operations defined in a software application's source code (entities) has nothing to do with the number of operations that it performed at run time (events).

While a basic type system is simple enough to implement, usually one should exploit an existing facility such as user-defined types in a programming language (e.g., C++ classes), description logic in an ontology language, or even a graph theory library to avoid building yet another instance of this abstraction. The most appropriate mechanism will depend on the idioms and interoperability needs of the particular environment being served.

For a complete discussion of this model, related work, and alternative approaches, please see Ref. [37].

6. Numeric data types

It is unfortunately common in scientific applications for all numeric values to be represented using floating-point numbers as the catch-all data type. While additional numeric types and arbitrary-precision arithmetic are widely implemented, their integration with scientific applications in general and quantity calculus in particular is inconsistent.

In addition to the basic types of floating point, signed integer, and unsigned integer, the following constructed types are frequently useful:

- Fixed-point numbers, specifically integers that are scaled by powers of 10, for exact representation of amounts of money and other quantities where the decimal places are inflexible;
- Rational numbers, represented as the fraction of two integers, for exact representation of values like 1/3 that floating-point numbers can only approximate; and
- Complex numbers ($a+bi$), represented using two numbers of any of the preceding types.

Basic types should be available in both the fixed sizes that are natively supported by the CPU and in arbitrary-precision form. The constructed types, in turn, should be usable with either fixed-size or arbitrary-precision representations.

Finally, it should be possible to enable and handle exceptions for numeric overflow, underflow, etc. The user should not need to instrument every numeric calculation at the application level to try to detect or prevent these errors, which are properly detected and reported at the arithmetic-logic level.

7. Conclusion

This short article described an architecture that addresses the complete set of functions for quantities and units in a simple and consistent fashion. Hopefully, this will encourage more convergent thinking about the functions and the underlying concepts so that the many disparate software implementations, present and future, will become more consistent with one another.

Acknowledgments

Thanks to Raghu Kacker, Paul Black, Barbara Guttman, Antonio Possolo, and CS&I reviewers for helpful review comments and suggestions.

Funding for the preparation of this article was provided by NIST.

References

[1] D. Flater, Architecture for Software-Assisted Quantity Calculus, NIST Technical Note 1943, National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899, 2016. https://doi.org/10.6028/NIST.TN.1943.
[2] J. de Boer, On the history of quantity calculus and the international system, Metrologia 31 (6) (1995) 405–429. https://doi.org/10.1088/0026-1394/31/6/001.
[3] Units—Wolfram language documentation, 2017. https://reference.wolfram.com/language/guide/Units.html.
[4] Ezunits, in Maxima 5.40.0 manual, 2017, http://maxima.sourceforge.net/docs/manual/maxima_56.html.
[5] Modelica SIunits library, 2009, http://goo.gl/0fEQ8N.
[6] Available units in LabVIEW—LabVIEW 2016 help, 2016. http://zone.ni.com/reference/en-XX/help/371361N-01/lvhowto/available_units_in_labview.
[7] V.F. Ochkov, V.I. Lehenkyi, E.A. Minaeva, Physical quantities, dimensions and units in mathematical packages, Mathematical Machines and Systems 1 (2009) 78–90. http://twt.mpei.ac.ru/ochkov/Units/QDUeng.pdf.
[8] GNU Units, 2017, https://www.gnu.org/software/units/.
[9] Unidata, UDUNITS package, 2017. https://www.unidata.ucar.edu/software/udunits/.
[10] E. Pebesma, T. Mailund, J. Hiebert, Measurement units in R, The R Journal 8 (2) (2016) 486–494. https://journal.r-project.org/archive/2016-2/pebesma-mailund-hiebert.pdf.
[11] K.C. Olbrich, Ruby-units, 2017, https://github.com/olbrich/ruby-units.
[12] M. Tanaka, Phys-units, 2016, https://github.com/masa16/phys-units.
[13] J.W. Lewis, Unitwise, 2017, https://github.com/joshwlewis/unitwise.
[14] M.C. Schabel, S. Watanabe, Boost.Units 1.1.0, 2017, http://www.boost.org/doc/libs/1_64_0/doc/html/boost_units.html.
[15] D. Delimarsky, Units of measure, in F# language reference, 2016, https://goo.gl/eOsNa5.
[16] S. Byrnes, Numericalunits, 2017, https://pypi.python.org/pypi/numericalunits.
[17] K. Fischer, et al., SIUnits, efficient unit-checked computation, 2017, https://github.com/Keno/SIUnits.jl.
[18] A. Keller, et al., Unitful.jl, physical quantities with arbitrary units, 2017, https://github.com/ajkeller34/Unitful.jl.
[19] International vocabulary of metrology—Basic and general concepts and associated terms (VIM), Joint Committee for Guides in Metrology, 3rd edition, JCGM 200:2012, http://www.bipm.org/en/publications/guides/vim.html.
[20] R. Dybkaer, An ontology on property for physical, chemical, and biological systems, IUPAC, 2009. https://doi.org/10.1351/978-87-990010-1-9.
[21] Unified Code for Units of Measure, rev. 2.0.1, 2014, http://unitsofmeasure.org/ucum.html.
[22] OASIS, Quantities and Units of Measure Ontology Standard, 2014, https://oasis-open.org/committees/quomos/.
[23] Units Markup Language (UnitsML), 2011, http://unitsml.nist.gov/, https://oasis-open.org/committees/unitsml/.
[24] QUDT.org, QUDT—Quantities, Units, Dimensions and Data Types Ontologies, 2017. http://www.qudt.org/.
[25] Quantities, Units, Dimensions, Values (QUDV), 2009. http://goo.gl/bIXclX.
[26] H. Rijgersberg, M. Wigham, J.L. Top, How semantics can improve engineering processes: A case of units of measure and quantities, Advanced Engineering Informatics 25 (2011) 276–287. https://doi.org/10.1016/j.aei.2010.07.008.
[27] BIPM, The International System of Units (SI), 8th edition, 2006. http://www.bipm.org/en/publications/si-brochure/.
[28] M.P. Foster, Quantities, units and computing, Computer Standards & Interfaces 35 (2013) 529–535. https://doi.org/10.1016/j.csi.2013.02.001.
[29] M.G. Morgan, M. Henrion, M. Small, Uncertainty: A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis, Cambridge University Press, 1990.
[30] Evaluation of measurement data—Supplement 1 to the "Guide to the expression of uncertainty in measurement"—Propagation of distributions using a Monte Carlo method, Joint Committee for Guides in Metrology, JCGM 101:2008, http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf.
[31] L.N. Trefethen, Computing numerically with functions instead of numbers, Communications of the ACM 58 (10) (2015) 91–97. https://doi.org/10.1145/2814847.
[32] J. Bornholt, T. Mytkowicz, K.S. McKinley, Uncertain⟨T⟩: A first-order type for uncertain data, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14), 2014, pp. 51–66. https://doi.org/10.1145/2541940.2541958.
[33] NIST Uncertainty Machine, 2017, http://uncertainty.nist.gov/.
[34] Wikipedia, List of uncertainty propagation software, 2017. https://en.wikipedia.org/wiki/List_of_uncertainty_propagation_software.
[35] J. Kogan, An alternative path to a new SI, Part 1: On quantities with dimension one, 2014. https://goo.gl/1FEPwN.
[36] P.J. Mohr, W.D. Phillips, Dimensionless units in the SI, Metrologia 52 (1) (2015) 40–47. https://doi.org/10.1088/0026-1394/52/1/40.
[37] D. Flater, Redressing grievances with the treatment of dimensionless quantities in SI, Measurement 109 (2017) 105–110. https://doi.org/10.1016/j.measurement.2017.05.043.