Measurement Data
(Archive Report)

William Kent [1]
Stephanie Leichner Janowski [2] **Bruce Hamilton** [1]
Dan Hepner [3]

Preface

The investigation captured in this archival snapshot was conducted sporadically from 1993 to 1995. It was sparked by HP's Computer Systems Division's intent to provide a model of dimensioned data in the OpenODB object-oriented database system, aimed at the needs of POSC (Petroleum Open Systems Consortium). Formal support waned,but several of us remained tantalized by the complexities and anomalies that lay just beneath the surface of such a nominally simple notion as dimensioned data. We met informally for a couple of years, progressing as best we could between the cracks of our "real" jobs, but it eventually became too difficult to sustain any real progress. After one last spurt as an HP Labs Grass-roots Basic Research Project in the summer of 1995, the effort just petered out.

These are the remains, a snapshot of our workbook frozen in mid-flight, with all the lumps and warts of unfinished work in progress. It's disorganized. This document was intended to be a repository from which various "real" documents can be extracted. It provided a framework in which all our stuff could appear, in some logical sequence. It contains material at all stages of completion, and at various levels of presentation for various audiences, from popularly tutorial to academically axiomatic.

Contents

---

1. Hewlett-Packard Laboratories
2. Hewlett-Packard Test and Measurement Support Division
3. Hewlett-Packard Computer Systems Organization

I PROLOGUE

1 Introduction

1.1 What's the Problem? (1)

- Computational systems don't provide enough support for measurement data. This is partly due to known problems in such areas as units conversion, accuracy and precision.
- Another reason, however, is an inadequate understanding of the fundamental nature of measurement, and the limitations of current theory, particularly in the area of dimensional analysis.
- "Measurement" covers a broad range of notions, some involving a units-based paradigm, some not.
- Dimensional analysis is insufficient for the general requirements of units-based measurement. There is a need for some new sorts of dimensions, and new sorts of relationships among dimensions, in order to meet the needs of more measurement applications. Much of this document addresses this need.
- As a prerequisite, such theory must rest on a sound foundation of concepts which clearly differentiate between physical quantities, units, and representations.
- Useful computational support for measurement data would include the ability to define new dimensions and units to the system. In order to understand exactly what domains are supportable by a unit-based computational facility, it's useful to have an axiomatic characterization of the behavior of units-based measurement.
- Beyond units-based measurement, it's useful to explore other measurement paradigms which might also be supported in computational systems.
- Finally, we examine the requirements for providing effective support for measurement data in computational systems.

1.2 What's the Problem? (2)

Travel planning involves arithmetic. How long will it take to drive from Tacoma to Tijuana? Let's see. We'll need about three days to drive from Tacoma to Los Angeles, then two more hours to San Diego, and Tijuana is maybe another fifty miles. Feed that data into a computer and you might get 3+2+50=55. Is that what you want? Not likely. The arithmetic is correct according to the computer's rules, but that's not what we need. We'd like to express the problem as "3 days + 2 hours + 50 miles". The clever computer shouldn't start by adding 3+2, but rather something a little smarter to get either 3.083 days or 74 hours. And then the computer should refuse to add that to 50 miles, knowing that you can't add a time to a distance.

Checking the legitimacy of an expression could be complicated. There are lots of dimensions: Length, Area, Volume, Time, Speed, Acceleration, Mass, Density, Force, Work, etc., etc., etc. It could take a book of rules to define what sorts of dimensional expressions can be added, subtracted, multiplied, divided, assigned, and compared. It's a little simpler if we ignore units. Whatever you can do with pounds can also be done with ounces, tons, grams, and kilograms, given the appropriate conversion factor. For checking legitimacy, it only matters that a weight is involved. Further simplification is then provided by dimensional analysis.

Classical dimensional analysism is elegantly simple. All dimensions can be expressed as multiplicative combinations of a few basic dimensions. Area is Length^2, Volume is Length^3, Speed is Length/Time, and Acceleration is Length/Time^2. Force is Mass times Acceleration, that is, Mass*Length/Time^2, and work is Force times Length (though many of us memorized the phrase "Force times Distance"), which is Mass*Length^2/Time^2.

The theory of dimensional analysis in twenty-five words or less: dimensions can be algebraically substituted, multiplied, and cancelled. Dimensionally equivalent terms can be combined, compared, and converted. Combination and comparison require common units.

Can you add a speed in miles per hour to a speed in feet per second? Yes, because they are the same sort of thing: dimensions of the form Length/Time. You just have to convert to common units first. There might be some quarrel as to which units to use, but the point is that it can be done and it makes sense. You can also compare a speed in miles per hour to a speed in feet per second, once you've converted to common units. And if you need something expressed in miles per hour though it's been computed in feet per second, is there a legitimate units conversion available? Yes, because they are dimensionally equivalent.

Can you add a speed to an acceleration? No, because one is Length/Time and the other is Length/Time^2. No amount of units conversion can make that a legitimate addition. No amount of units conversion can make sense out of comparing the two. There is no legitimate units conversion from one to the other.

If you divide an area by a length, and divide that by a time value, can you add the resulting term to the result of multiplying an acceleration by a time value? Yes, because each term works out to be Length/Time, hence both are speeds, and you can always add the same sorts of things so long as you're careful to regularize the units.

You can learn this elegant theory of dimensional analysis in about a minute, and it solves a lot of problems. Very useful type systems based on dimensional analysis could be designed for programming languages to support dimensioned data. ***[Maybe say a few words about potential difficulties with regard to units naming, recognition, and defaults; conversion algorithms; accuracy and precision; syntax for expressions and declarations; and whatever else might be relevant at that level.]***

But this simplicity comes with the usual sort of price. As good old Albert said, things should be as simple as possible, but not more. ***[Verify the source, and the wording.]***

Fuel consumption in gallons per mile is a volume divided by a length, yielding Length^2, which is Area. Is fuel consumption really the same sort of thing as area? Is adding a fuel consumption to an area any more sensible than adding an area to a volume?

The coverage of paint, if measured in gallons per square foot (Volume/Area), would be dimensionally equivalent to Length. Does that make sense? And is the productivity of a lumber mill or licorice factory, measured in feet per day, really the same concept as speed, even though both are Length/Time?

With a little bit of high school physics you would know that torque is a force multiplied by a length — and you would also know that torque is not the same thing as work, also force times length.

So we have a host of dimensions which are "dimensionally equivalent" without really being semantically equivalent. They aren't compatible with each other, even though dimensional analysis says they are.

The opposite problem exists, too. There are dimensions which are semantically equivalent, and should be compatible, even though they aren't dimensionally equivalent. If I have a cup of salt and you have a pound of salt, does it make sense to ask who has more salt? Of course it does; you and I can make that comparison. Does it make sense to ask how much salt we would have if we added mine to yours? Of course. But dimensional analysis wouldn't allow us to compare or add a volume and a mass, even though both are notions of "amount".

If we each dissolved our salt in a quart of water, could we figure out which solution is more concentrated? Of course, even though mine is initially expressed as Volume/Volume and yours as Mass/Volume. Again, though these are both “concentrations”, dimensional analysis doesn’t recognize their commonality.

If we each paid a dollar for our salt, we could figure out whose was more expensive. There’s a common concept of “unit cost”, though they aren’t dimensionally equivalent.

Of course, there’s a catch here. The relationship between a cup and a quart, or between a gram and a pound, is the same no matter what kind of stuff we’re talking about. But in order to compare my cup with your pound we have to know whether we’re talking about salt, sand, water, mercury, or something else. The correspondence between a volume and a mass depends on the density of the stuff involved. But once we’ve determined that, once we’ve established a context in which the density of stuff is constant, we can treat “amount” as a dimension which can be expressed as mass or volume interchangeably in much the same way as units of volume are interchangeable. This has important applications in many industrial settings, and even in the medical context, where medication dosages are routinely converted between different dimensional forms. In general, some incompatible dimensions can be made compatible by fixing certain conversion parameters for a particular context.

Dimensional analysis does strange things with “dimensionless” quantities. If I dissolve my cup of salt in a quart of water, I might report the concentration in terms of Volume/Volume. If you dissolve your pound of salt in ten pounds of water, you might express the concentration in terms of Mass/Mass. In each case the dimensions cancel out, leaving us with dimensionless quantities, which are considered to be dimensionally equivalent. But the slope of a ramp, which is Length/Length (e.g., inches/foot), and the accuracy of a clock, which is Time/Time (e.g., seconds/day) are also dimensionless, and dimensional analysis treats these all as being compatible with each other. Our two salt-water concentrations, the slope, and the clock accuracy can all be freely compared and added to each other. Furthermore, angle is often equated with Length/Length, making it yet another dimensionless quantity compatible with all of these.

Can such problems be solved? Yes. At what price? Complexity. Traditional dimensional analysis has exactly one sort of relationship among dimensions: any dimension can be uniquely expressed as a multiplicative combination of basic dimensions. Two new sorts of relationships address many of these other problems: a dimension can be either a “specialization” or a “generalization” of another. What we mean by those terms will be explained at great length later on.

Other problems suggest other new sorts of relationships among dimensions...

“And now, the world-wide weather report. The temperature in Stockholm reached 40 °C today. New York City was just as hot, reporting 104 °F. Stockholm’s temperature has jumped by 40 °C in just one day, which also matches the one-day jump of 72 °F suffered by New Yorkers.”

You think that weather’s amazing? How about those amazing comparisons. 40 °C is the same as 104 °F, and 40 °C is also the same as 72 °F. What’s going on? Aren’t we talking about the same sort of physical quantity? Apparently not. In one case we’re reporting temperature points: how hot is it? The “hotness” measured as 40 °C is the same as the hotness measured as 104 °F. The other case deals with temperature intervals: how much has it changed? The change measured as 40 °C is the same as the change measured as 72 °F. Thus temperature is not one but two dimensional domains, related to each other as a “point-interval” pair. This is another new sort of relationship among dimensions that we’ll describe later.

Other ambivalent behaviors can also be resolved by distinguishing between similar domains. Is an angle of 400° the same as an angle of 40°? Yes, if the domain is circular angles, but not if the domain is rotational angles. Are there such things as negative weights? Not if you’re talking about the weights of things, but yes in the case of buoyancies. Lengths similarly occur in two domains which do and don't admit negative values.

These treatments of dimensioned data will be realized in the type systems of programming languages. Robust systems will not only provide support for a particular set of dimensions, but will allow new dimensions to be defined as well. What determines whether something can be legitimately defined as a dimension to be managed by such computational facilities? Could things like color, intelligence, efficiency, productivity, reliability, ... be handled? Why or why not? This leads to an exploration of the formal foundations of units-based dimensioned data.

Before elaborating these new ideas about dimensioned data, we need to be clear enough on the basic concepts. Is "10 pounds" a weight or a price? Is it the same thing as "10 lbs"? As "160 ounces"? To sort these out, we distinguish between physical quantities, units, and representations, as explained later. Briefly, a physical quantity such as a certain weight is a distinct abstract thing, quite apart from the various expressions of the form "10 pounds" or "160 ounces" which might be used to express it. This physical quantity is much like the idea of a person being a single thing, distinct from the various names, employee numbers, social security numbers, and other constructs used to represent him. Then we recognize units, used for measurements of certain kinds of physical quantities, as being abstractions apart from their names. For example, there are two distinct units, one measuring weight and the other measuring money, even though both are called "pounds". Conversely, one of these units might have several names such as "pounds" and "lbs". Finally, representation has to do with the formats of text strings expressing some sorts of numbers and unit names.

1.3 What's the Problem? (3)

Dimensioned data occurs in computing systems almost everywhere, but is poorly supported. Computing systems routinely:

- Convert among compatible representations such as integer and floating point:
  - 6+3.1
  - 6 eq 3.1?
  - (float) x ← (int) 6.
- Report errors when asked to convert among incompatible representations:
  - 6+"ABC"
  - 6 eq "ABC"?
  - (short) x ← 10000000000.
- Perform scale conversion transparently and efficiently when needed. In the usual floating point representation, the sum 10 000.1 + 0.07 is computed by first scaling one of the numbers until the two exponents are equal, then performing the addition (and possibly normalizing the result). The equivalent operation on dimensioned data is the same, with "exponent" replaced by "unit."
- Allow users to define new types, and make specializations and collections of existing types.

But the same systems which have built-in support for expressions like 10 000.1 + 0.07 offer no support for expressions like 2 miles ÷ 15 minutes. If such support were available, it would offer the same benefits as the support for number types: make data more self-explanatory, save time, eliminate error-prone tasks, catch errors in the remaining tasks, and improve interchange of data across systems and between systems and people. In a database engine, such support would save bandwidth and ease the load on client systems.

1.4 What's the Problem? (4)

Dimensioned data is important in many computer applications, and will become even more important in emerging application domains such as health care and the environment. The most familiar computational aspects involve units conversion and dimensional analysis. Behind these deceptively simple aspects lies a remarkably confusing set of issues concerning the appropriateness and meaning of various operations. Sorting out these issues is essential to the design of appropriate facilities in programming languages and information modeling.

Even the scope and boundaries of the topic are unclear. Measurement generally involves assignment of a symbolic representation to a physical quantity in a way that communicates certain information and supports certain operations. "Physical quantity" is a very loose term, covering obviously physical things like length and weight as well as not so physical things like intelligence, efficiency, and beauty. Sometimes we'll refer to these more vaguely as "phenomena".

Several aspects of dimensioned data need to be distinguished:

- The abstract phenomenon being characterized.
- The corresponding symbolic representations.
- The abstract paradigm for assigning a representation to the phenomenon, e.g., judging, testing, polling, measuring, etc.
- The real process for assigning a representation, involving such aspects as accuracy, precision, and repeatability, as well as such auxiliary mechanisms such as averaging of multiple readings.

The symbolic representation is often numeric, but letter grades in school and letter compass headings such as NNE may also be considered to be dimensioned data.

The representation may simply convey a relative order, such as first, second, and third place, or gold, silver, and bronze medals. Very often the information has the sense of how big or how much, but not always. Some phenomena don't involve a sense of order or bigness at all, such as locations or directions.

There are various paradigms for assigning symbolic representations to phenomena, which may or may not all be considered "measurement". At one extreme are purely subjective judgments, as in grading essays, or in judging Olympic events, beauty contests, and livestock competitions at the county fair. A bit less subjective are polling and testing procedures which simply count things, like people's preferences or the number of correct answers on a test. Some paradigms are based on more complicated computations, as in batting averages and quarterback ratings.

In its narrowest sense, the term "measurement" involves a unit, which is a particular instance of the measurement domain. Any thing in the domain is then expressible as being equal to k of the unit things. The value of k is clearly different for different choices of the unit thing.

Symbolic representations have behaviors of their own, such as ordering among letters and numbers, and arithmetic among numbers. Such behaviors may or may not correctly model the behaviors of the underlying phenomena. Ordering often makes sense: it's common sense that a rock whose weight is a bigger number is a heavier rock (assuming common units). On the other hand, compass headings don't have order: NE is not bigger or smaller than SE, nor is a direction of 30° bigger or smaller than a direction of 60°. And it is at least arguable whether a higher IQ number implies a smarter person, or a higher letter grade implies a better performance, or a higher price implies a greater value.

Arithmetic can be similarly misleading. Just because numbers can be added and subtracted doesn't mean that these operations are meaningful for measurements. Weights are well-behaved: putting two rocks on a scale yields a measurement which is the sum of the individual weights, so it is sensible to add weight measurements (in consistent units, of course). But it doesn't make sense to add temperatures, or numeric compass headings, even though they are numbers. If the direction of travel of two ships is 30° and 60°, what does it mean to add those two numbers? (Curious coincidence that temperatures and headings are both measured in degrees.)

Other arithmetic behaviors may also fail to correspond to the underlying phenomena. Adding a positive number to a positive number yields a larger number, but such ordering doesn't hold for circular angles: adding 200° and 200° yields an angle of 40°. Adding the same angle to two angles can invert the order of the results: $50° < 70°$ but $300°+50° > 300°+70°$, since $350° > 10°$.

Subtraction doesn't always make sense, either. While numbers are closed under subtraction, sometimes yielding negative numbers, it doesn't follow that subtracting a large weight from a small weight yields a weight.

Even simple counting can be anomalous. With angles, counting does not lead to a purely monotonic sequence. As we keep accumulating one-degree increments, we eventually come back to angles we've encountered before. Among other things, this implies that the "measure" of an angle is not unique: $x=x+360i$ for any integer i.

Which operations with dimensioned expressions are valid, and what their results signify, depends on the type of underlying phenomena involved. Weights, temperatures, angles, velocities, and intelligence all behave differently. Even though measurements are expressed as numbers, we can't assume that all properties of numbers automatically apply in these domains. In fact, it's necessary to reason the other way. We need to independently establish whether certain principles hold in the underlying domain in order to justify applying various arithmetic procedures to the measurements. In other words, we need to understand the extent to which we can or cannot reason about a measurement domain by reasoning about the measurement numbers.

Simply put, you can get into trouble if you just assume that properties of numbers automatically imply corresponding behaviors in the underlying domain.

Subsequent sections identify a set of principles (sometimes in the form of axioms) which may or may not hold in various measurement domains. If an axiom doesn't hold in a physical domain, then arithmetic operations which depend on that axiom may not be meaningfully performed on measurements of that domain.

Such principles serve several purposes. First of all, they can be grouped into subsets to characterize different sorts of measurement systems. Such sets of principles precisely and generically describe different levels of support which can be provided in a language.

Further, the principles provide an analytical tool, helping to determine precisely whether and how a given domain can be subjected to measurement. The analysis can expose ambiguity and fuzziness of concept, leading sometimes to differentiation of similar domains, sometimes to clarification of exactly how measurement principles can be applied. The principles also illuminate why certain domains such as color or beauty are difficult to measure.

1.5 What is Measurement? (1)

What is measurement? It's hard to say with great precision; every generalization seems to admit a counterexample.

In its most general sense, measurement is a form of naming, assigning a symbolic representation to some concept so that we can communicate it. It enables us to say that something is 10 feet long, weighs 100 pounds, and is heading northeast at 100 miles per hour.

How does measurement differ from naming? We usually think of measurement as yielding numbers, but there are exceptions: school grades, compass headings of the form "NE" (or northeast), and perhaps the names of colors — if we agree that color constitutes a measured thing.

Naming is generally associated with identifying concrete things such as people or chemical elements, while measurements identify properties of concrete things, such as their weight, speed, direction, and so on.

Measurement provides representations which distinguish members of a set, such as the various weights, or the various directions. In this respect measurement is still similar to naming, as with people or chemical elements.

Well, some measurements are like names in that they identify specific values, but other measurements denote relative performance in some trial. A batting average relates to absolute data about a batter's hits and times at bat, while first, second, and third place in a race refer to relative performance as compared to whoever else was competing at the time. Score on a standard test vs. grading on a curve.

Some measurements are relative but objective, while others are subjective. School grades?

Naming and measurement are both used to communicate information. Measurement, in addition, facilitates certain operations on the information. The most elementary of these is comparison: higher numbers imply "more" of the quality; earlier letters in the alphabet imply "greater" performance. Thus measured things have an ordering property: one is "larger" than another.

Most things, that is; there are exceptions. Compass headings, for example, don't have such sense of comparative order. East is not bigger or smaller than northeast. Even when expressed as numbers, a heading of 90° is not intrinsically bigger than a heading of 45°. One may make a bigger angle with the north-south line than the other, but the heading is not bigger.

Compass headings do, however, have a sense of "betweenness" — in most cases.

A thing is a dimension only by convention. In some subdomains of physics, the speed of light is defined to be 1, and speeds do not appear in equations. Some domains assume a two-dimensional world, and have no notion for height. The laws of electricity can be restated in terms of numbers of electrons, rather than charge. The decision to regard a thing as a dimension, and to measure and specify its instances, is a matter of convenience. Similarly, the notion of measurement is based on a conventional measurement procedure. Pulse width is usually measured between the 10% and 90% amplitude points, while Length is measured between extrema. Length is measured using straight-line distance (but sometimes great circle distance), weight is measured in a frame of reference which is stationary with respect to the earth's surface (as opposed to free fall, for example). Finally, the preciseness of the thing being measured is a matter of both convention and context. Statements of the distance to Los Angeles rarely specify whether they mean the nearest city limit, city hall, a marker at city hall, the center of the marker at a particular date, etc.; but sometimes they do specify those things. We assume that those issues have been settled for the context in which measurements are made, and discuss what is required to communicate and manipulate them.

1.6 What is Measurement? (2)

Measurement communicates and computes.

Measurement provides a way of systematically naming certain kinds of things when there are too many of them for individual names. Imagine that we could only refer to a particular day as "Bill Kent's birthday" or "the day Bill's first cat died" or "the day Farmer Brown's barn burned down". We'd have a hard time inventing unique names for every single day. Even with unique names, we'd probably have lots of names for any given day, though most such names wouldn't be recognized by many people. And we wouldn't know very much about days named in such a way.

Imagine trying to name weights or lengths that way. As wide as Bill's hand. As far as the distance between New York and Los Angeles. As tall as the Empire State Building. As heavy as Bill when he was born. As heavy as Farmer Brown's prize bull.

Measurement gives us a sense of certain relationships among things. It tells whether one is less or more than the other in some way, or if they happen to be the same. And it tells us whether they are close together or far apart. It gives us a way to express what we want or need for a certain situation, and to judge whether something satisfies our needs.

Measurement provides a way of correlating things with numbers, so that we can assume they behave in certain familiar ways. We know what it means to add, subtract, multiply, divide, and compare numbers, so we can do the same sorts of things with the things we measure. We get a sense of what it means to add two weights together, or to figure out the ratio between them. The notion that "this stone is half as heavy as that one" means something to us in precise, numerical terms.

We can't measure any old thing we want. We can only measure things with numbers *if* they behave like numbers to begin with. If we didn't start out sensing (and agreeing) that two of these stones weigh the same as that one, then it wouldn't make sense to do arithmetic with the measurements. And just because we can assign numbers to things, it doesn't follow that they are measurements. Lots of things have serial numbers, like employees and cars and television sets, but it doesn't make much sense to add or divide these numbers. They aren't measurements.

Suppose you're trying to figure out a way to measure certain things. You should first check whether those things have any behavior of their own that's similar to the way numbers behave. Is there a sense of order that lets you say one is less than the other in some way? Is there some way of combining two of those things in a way that resembles addition? Can you imagine some sort of numeric ratio between two of them, to support a notion of dividing one by the other? Such questions might explain in a fundamental way why it's so hard to measure intelligence, beauty, color, efficiency, proficiency, productivity, and a host of other things we're anxious to measure.

By these criteria, temperature is clearly not in itself a well-behaved units-based interval domain. At most we have a sense of order by which we can compare the magnitude of two temperatures or temperature intervals. My coffee is hotter than yours. His coffee is even more hotter than yours than mine is. But we can't really add them. Do we have any real intuition that the temperature of my coffee and the temperature of your coffee combine to be the temperature of his coffee? Could I judge that the warmth of your coffee is about the same as the difference between his and mine? Not at all.

What are the numbers of temperature? A measure of a very indirect phenomenon. We happen to have noticed that, for most substances, when we agree that a sample of it has become warmer we also observe that it occupies more volume. When a certain sample of mercury or alcohol in a tube feels warmer to us, it also rises higher in the tube. Is it twice as warm when it's twice as high? We don't know that. Has the warmth changed by twice as much when the level rises twice as much? We don't know that either.

We don't really know that there is a "unit" of temperature. Do we have any independent judgement that if his coffee is one degree hotter than mine, and mine is one degree hotter than yours, that the difference in warmth is exactly the same in both cases? No. All we know is that the distince the mercury moves in its tube is the same. The "unit" is really a unit of distance. Two degrees and three degrees mean that the distance in the tube is two or three times as the distance for one degree. Not that we know that something is essentially twice or three times as much in warmth.

[I think we've also discussed the following. As a first approximation, specific heat suggests that there is in fact some basis for additive behavior of temperature. There is a sense in which it can be demonstrated that the amount of heat required to raise the temperature of a given substance by one degree is constant. This can be done in a non-circular way, which doesn't depend on the artificial nature of temperature measurement. However, I believe that this constant is not precisely constant, i.e., that the amount of heat required to raise a substance's temperature from 10 to 11 degrees is not quite exactly the same as the heat required to raise its temperature from 100 to 101 degrees (assuming no state change in that temperature range). Is that true, then this adds support to my thesis about the nature of temperature measurement, namely that temperature doesn't really correspond to an additive behavior.]

1.7 Goals of Measurement Systems

In general we think of a measurement system as something involving units, providing the ability to do units conversion and to validate and evaluate expressions involving units. In broadest terms, a *measurement domain* contains a set of *physical quantities* we wish to map into symbolic representations which convey information and support certain operations, in a manner consistent with our intuitive understanding of behaviors in the domain. The term "measurement" is used to mean either the mapping process or the resulting symbolic representation. A measurement system thus provides:

- Symbolic notation: a mapping from the abstractions in some domain (such as weights or performance in a course) into some symbolic representation.
- Operations: operations on the symbolic representations which should mimic behaviors of the abstractions being represented. For example, addition of numbers representing weights should preserve the physical behavior of weights being combined.

1.7.1 Symbolic Representations

The simple business of assigning symbolic representations to the abstractions in some domain is sometimes a major challenge in itself.

We usually think of measurement systems as dealing with numbers, but non-numeric representations are sometimes used. School grades and letter compass points (e.g., SSW) are examples.

Some measurements require a vector of representations rather than a simple number.Velocity is the most familiar example, requiring two values to express the speed and heading. Other examples include location in n-dimensional space, and colors expressed in terms of hue, saturation, and value ***(explore the behavior of those three as scalar measurements)***. This category doesn't include "additive" notations such as "five pounds, three ounces" which can trivially be transformed into a single number. True vector measurements don't have a single-number equivalent.

Even scalar (single-number) measurements can be difficult to quantify. Thus the "soft" sciences have problems trying to assign measures in such domains as intelligence, performance in various senses, usability, quality, productivity, efficiency, wealth, net worth, cost of ownership, and various other economic indicators. Similar issues also arise with beauty contests, movie and restaurant ratings, and judging of Olympic events. Symbolic representation can often be misleading, implying greater precision than actually exists, and sometimes even giving rise to incorrect comparisons. Does higher IQ really imply greater intelligence?

In this section we focus primarily on well-defined scalar measurement systems. The other issues should be dealt with in other sections of the main paper.

1.7.2 Operations

Various combinations of the following operations with symbolic representations are supported in various measurement domains:

- Communication. This simple facility should not be overlooked, particularly in non-numeric representations. Letter grades and compass headings convey information. So do all the other measurement notations. In the case of letter grades or economic status indicators, this may not always be self-evident. More generally, we expect numeric representations to convey some intuitive sense of magnitude, i. e., how big, how much.
- Equality comparison. We generally expect that if the symbolic representations match, then the abstractions being represented are the same in some sense. (The converse needn't hold; units

conversion presumes that equal quantities may have different representations.) Even this could be problematic, as with school grades. Exactly what does it mean to give the same grade to two students in two courses at two schools?

- Order comparison. If one symbolic representation is “larger” than another, we expect the corresponding underlying abstraction to be larger than the other. Problems again, as noted with IQ’s and school grades.
- Combination within a domain. We very often wish to add the measurements of two weights, or of two lengths. This is intended to mimic the behavior of combining the underlying abstractions. Adding the measurements of two weights should yield a result that’s consistent with the effect of putting two rocks on one pan of a balance scale. Other operations such as subtraction, multiplication, and division should make similar sense.
- Combinations across domains, as in dividing Distance by Time to get Speed.

1.8 This Paper

1.8.1 Goals

This paper explores the functionality required for a computer system supporting *dimensioned data*, that is, data about physical quantities, including their dimensions and units. It advances a theory which covers not only well-behaved physical quantities like mass and length, but also ill-behaved quantities like temperature, concentration, and “dimensionless” quantities. The theory is extensible to user-defined dimensions and units. It is motivated by the process of measuring physical quantities, and therefore touches on two related issues: the context surrounding measurements, and the precision and accuracy of measurements.

Previous efforts to support dimensioned data have been weak and incomplete ***[refs]***. We hypothesize that the reason for this is the lack of an underlying theory. This paper fills that gap. An additional benefit is that a thorough exposition of the subject is interesting in its own right.

Thus a major contribution of this paper is a significant enhancement of the theory of dimensioned data. Another important contribution is the specification of a type system for programming languages, first in language-neutral terms and then in terms of various specific language bindings. ***[Will we ever get around to it?]***

The term “Dimensional Analysis” [9, 25] is sometimes used in this context, but it applies more properly to the use of dimensions to discover some of the equations which model physical problems. That is not a subject of this paper, and we don’t use that meaning of the term here.

1.8.2 Scope

We focus more on new areas not being investigated elsewhere. Other known problem areas, e.g., units conversion, accuracy and precision, are being investigated elsewhere and get less attention from us.

We cover as many aspects of dimensions and units as possible, and as many aspects of user-visible functionality as possible. Our treatment is more thorough than any previous work, although we exclude some areas. People in various times and contexts have chosen to regard different things as dimensions, and to describe the world along those dimensions. We sidestep such considerations; if someone wants to define Earth, Air, Fire and Water as dimensions, he can do that. We need to know only whether two dimensions are equal, and whether two units are equal. This allows us to treat “6 mm + 11 mm” one way, “6 mm + 11 inches” a different way, and “6 mm + 11 kg” in a third way.

Similarly, when people quantify the world along some dimension, they may apply different forms of arithmetic to the resulting numbers. R1+R2 describes the resistance of R1 and R2 connected in series, but T1+T2 does not describe the temperature when two fluids at temperatures T1 and T2 are mixed together. That temperature *can* be computed, however, and the computation makes use of the usual meanings of addition, subtraction, multiplication and division. In our work, therefore, we don't try to change the meanings of arithmetic, vector mathematics, statistics, etc.; we merely study their application to physical quantities.

Having said that addition, subtraction, etc., mean the usual thing, we need to characterize the kinds of data for which this behavior gives intuitive results. We do that in Section 34.7. Informally, they are data which represent counts of some unit along some dimension.[1] This category excludes many other ways of quantifying the world, such as golf scores, wire gauges, pineapple sizes (the number of pineapples that fit in a standard box), preference rankings, intelligence test scores, beauty contest scores (number of judges who rank the candidate first), etc. These are interesting (some are discussed in [8,13]), but their arithmetic is ill-behaved, and in practice these quantifications are rarely used for more than ranking or binning.

The well-known rules for dimensions and units, such as in the resistance example cited above, are simple. They are taught in high school, implemented for the length dimension in draftsmen's calculators, and implemented for a moderate set of dimensions in the HP-48GX handheld calculator. We refer to such treatments as *basic dimensional support*. There are also unusual cases, however, and they are unusual in many different ways. Some of the issues we will discuss in this paper, and which our theory will address, are enumerated in Section 2.

We cover those issues in this paper, but we do not try to cover more than requirements. We do not attempt to design a system, to choose which requirements have to be met to satisfy some market segment, nor to say how to weave dimensions and units into any existing product. We occasionally comment on likely interactions with programming languages, but we do not recommend any particular language. We occasionally comment on efficiency or other implementation issues, but we raise the issues only because they are interesting; we make no claim about how important they are, and we do not claim to raise all important issues.

1.8.3 Approach

Our general approach to such problems is to introduce more "dimensions", and more complex ways in which dimensions interrelate as compared with traditional dimensional analysis. In traditional dimensional analysis the base dimensions are orthogonally independent, such that any dimension can be uniquely represented as a multiplicative combination of base dimensions. This scheme is simple and elegant, but at the price of admitting the sorts of problems outlined above. We are essentially reassessing this tradeoff, exploring how much more useful functionality can be achieved at the price of increased complexity.

Some of the new sorts of inter-dimension relationships:

- Specialization.
- Generalization.
- Point/interval sort of correspondence.
- However we solve the multiple angle domains.
- Signed vs. unsigned. (Might be same mechanism as angle family.)
- Subtype (if different from prior solutions.)

1.8.4 Our Contribution to Dimensional Support

We draw strict distinctions among quantities, dimensions, units, conversions, and between those things and their representations. No two of those things lie in a 1:1 correspondence, and thus we are

1. Actually, we consider any isomorphism to such a count.

able to capture a certain amount of clarity and representational power which is missing from papers which confuse or combine them. We are also able to discuss issues which aren't considered in other work. We relax the assumption of a 1:1 correspondence between dimensions and their canonical representations (see Section 3.3), and explore both the 1:n case and the n:1 case. We deal with non-trivial dimensions and units, such as date, temperature, and angle, as well as time-varying and "dimensionless" quantities. We proceed without assuming an implementation in any particular programming language, though we note issues of representation and supporting functionality as they arise.

1.8.5 Prior Work

The notion of using dimensional analysis to detect errors in scientific and engineering computation has been used since the 1920's [9, 10]. With the advent of programming languages supporting user-defined types, various researchers [6, 7, 11, 16, 17, 18, 19, 20, 22, 26, 27, 29] have proposed building support for dimensions into Pascal, Ada, C++, ML, and relational database management systems, both to catch errors and to perform automatic unit conversions.

Most of this prior work has been limited by what was achievable through small extensions to the type system of the programming language under consideration. To varying degrees, all of it has mixed the notions of dimension, unit, conversion and representation, and has given uneven treatment to such anomalous cases as temperature, logarithmic units, dimensionless quantities and time-varying conversions. [27] escapes the limitations of programming languages, but intermixes dimension, unit and conversion almost completely. All assume a 1:1 correspondence between a dimension and its representation in terms of reference dimensions.

2 The Skeletons in the Closet

This is a reference list illustrating the problems and anomalies we will be addressing. The intent is that all situations listed here are discussed, and hopefully solved, in the document, and that all problems and anomalies addressed in the document are listed here.

> ***>>ToDo 1:** Check that all sample problems are covered in text.*
>
> ***[Oh well, the ordering and grouping seem to be breaking down. Need to fix it. It's also far from complete.]***

2.1 Domain Semantics

- Computational anomalies:
  - Some angles are measured modulo $\pi/2$, some modulo π, some modulo 2π, and some are simply measured cumulatively, not modulo anything (it does make sense to wrap a line around a capstan for 5π radians). Other angles, denoting directional headings, might not behave quite like measurements at all.

    [Do we really have examples of angles measured modulo $\pi/2$ or π? Interior angles [Section 7.1.3] have the property that $\pi+x=\pi-x$, but that's not modulo arithmetic.]

    Is 400° the same as 40°?

    Is a heading of 90° (East) the same as an angle of 90°?

    Is East less than West?

    Is 90° < 270°?

    Does East + East = South?
  - Is the difference of two temperatures still a temperature? Really in the same sense? A temperature of 5°C is 41°F, while a temperature difference of 5°C is 9°F.

- It makes sense to add, multiply and divide temperatures, but only in the Kelvin scale. Temperatures in any scale can be subtracted.
- It doesn't make sense to add, multiply or divide dates in any scale, but dates in any scale can be subtracted. The same is true of locations. However, the differences between dates, or locations, are not dates or locations.
- Time is the same kind of thing as date, but it is also sometimes measured modulo 24 hours, or modulo 12 hours. Time and date are usually expressed separately.
- Measurement of time is dependent on time zones and time of year, as though there were 24 parallel times lines (plus one for "absolute" time) whose positions shift twice a year. Events occurring synchronously might occur at different times, even on different dates. Events occurring at exactly the same date and time might not occur synchronously.
- Some clocks start and stop, and time lines might be discontinuous with respect to each other. Touchdowns scored at 5 and 6 minutes into the first quarter may have occurred at 1:15 and 1:20 in the afternoon.
- Is the difference of two weights always a weight? Even if negative? The same can be asked about any dimension.
- Equality

  The time domain is really problematic. When is a month a month? January and February are both months; are they equal? How many days in a month? Are any two years equal? What does it mean to add a month or a year to a given interval? And let's not forget about the mythical 30-day month and 360-day year.

  Lots of domains have equality: weights, temperatures, spatial locations, colors. Lots of domains don't, such as beauty and intelligence. (Don't confuse equality of measurement with equality of physical quantities. Equal IQ doesn't really mean equal intelligence.)

• Which of the following are well-behaved measurement domains? Why or why not? What are their significant properties?

- Locations in space.
- Headings.
- Money values.
- Intelligence.
- Beauty.
- Athletic performance of various sorts (batting average, slugging average, team standings, quarterback rating, Olympic event scoring, boxing results, gold/silver/bronze medals, ...).
- Productivity, efficiency, usability, network behaviors, ...
- Complexity (in the context of software metrics for application development).
- School performance (test scores, class grades, GPA, standing in class).
- Restaurant and movie ratings.
- Color temperature.
- Jitter ***(ask Bruce and Dan)***.
- pH.
- dB.

- Richter scale.
- Jolt.
- Hardness.
- Elasticity.
- Relative humidity (a ratio of concentrations).
- ...

- How do we characterize the notion of a "well-behaved measurement domain"? Are there different sorts, characterized by different behaviors? Thus:
  - How many different "angle" domains are there?
  - How many different time domains?
  - Are time and date really different domains?
  - How do we deal with weight and mass?

  [Hypothesis: major division of measurement domains/paradigms into units-based or not. Each one then has further subdivisions.]

- Angles have dimensions of Length/Length; so do strain and length error. Clock rate error has dimensions of Time/Time, and concentration has dimensions of Volume/Volume and Mass/Mass. The conventional theory states that these are all equivalent, and may freely be added, compared and assigned to one another.
- Vector physical quantities (like velocity) are well-modelled by many of the usual vector operations, but outer product is meaningless, and multiplication by a scalar yields the expected result in only one coordinate system.
- Torque and Work both have dimensions of Force $\times$Length, but are not the same thing. The conventional theory says they are equivalent.
- The conventional theory says that all lengths are equivalent, so that, for example, a length may be added to a width (yielding what?), and Height^2 =Area.
- The conventional theory makes certain dimensions incompatible when they shouldn't be, e.g., concentrations expressed as Volume/Volume or Weight/Volume.
- Instances of physical quantities:

  It is necessary to be able to identify, refer to, and differentiate various instances of physical quantities. This often takes the form of some property of some "subject" under certain circumstances, such as the temperature at which water boils at a specified pressure, the weight of Bill Kent on the planet Earth at 8 AM on May 19, 1994, and the distance between New York and San Francisco.

  Such references are often somewhat ambiguous, leading to issues of precision and accuracy. Bill's weight may or may not include his clothing. The distance between New York and San Francisco may be measured between various precise points. It might not even be straight-line distance; it might be airline miles or driving miles.

  We sometimes visualize an angle in terms of two lines meeting at a point. Are we always sure that a picture like < means the inside angle? How do we then refer to an angle greater than 180°?

- Reality?

It may also be useful to clarify whether a domain need be limited to quantities which can exist in reality. Dimensional systems are used not only for measuring real phenomena, but also for describing hypothetical or fantastic situations. Does the speed domain include speeds greater than the speed of light? Should 2c be an invalid speed expression? (Ask your friendly neighborhood physicist about tachyons.) Does the energy domain include half a quantum? Then perhaps it might also be meaningful in some contexts to contemplate negative masses, and negative temperatures.

- Color

  The kinds of combining operations we might imagine for colors include mixing crayons on paper, mixing lights, or mixing filters. I can't image any sort of combining operation and any sort of color such that repeatedly combining that color with itself will eventually reach any arbitrary color. Is there an inherent reason for this? Is that because it is a vector quantity? (No, that's probably not the whole reason.)

  Furthermore, I find it hard to even imagine a combining operation under which combining a color with itself yields anything other than the same color, except for an operation that annihilates the color to yield black.

  Explore the behavior of color as a vector of hue, saturation, value (HSV). Does it still satisfy our intuition that all vector quantities can be combined by vector addition? Also explore the behavior of each of those components as a scalar measurement domain. Actually, is color an HSV vector, or a vector of such vectors, one for each primary color?

- Sounds and Music

  Note names in music constitute nomenclature for cyclic values, just like days of the week or months.

  We may want to distinguish between combinations of sounds (directly experienced physical phenomena) and combination of frequencies (attributes of sounds). There are several analogies here with light (color), including the distinction between pure and mixed frequencies. Combining sounds, i.e., the result of playing two sounds together, is not simply described as a sum of frequencies.

- Temperature

  The more we say about this stuff, the less I understand how temperature works. Little of the preceding analysis seems to apply to temperature.

  Perhaps the only way to rationalize temperature is to be very simplistic and consider the coercions that go on in the measurement process. What is a degree of temperature? It's a distance! What we look at on a thermometer is how far something has moved up or down in the tube. The coercion is via the thermal expansion of the substance in the tube. We don't really know anything about the linearity of heating, or expansion. We simply mark the tube in uniform distance increments. We don't know if it takes twice as much heat to make the fluid move up twice as many notches. ***[There seems to be some dependence here on the uniformity of a substance's specific heat at different temperatures.]***

  So, temperature makes sense if we consider it a coercion to distance. Our naive intuitions about temperature are really based on the behavior of distance.

  As with volumes, mixing of substances doesn't give the right metaphor for combining temperatures. Ignore chemical reactions, i.e., let's assume we are dealing with kettles of water. They act somewhat like the self-nils of color: mixing two kettles at the same temperature yields something of the same temperature again. More generally, mixing two kettles does not yield something bigger than either one separately, but rather something in between: $x<y \Rightarrow x<x\#y<y$.

A better metaphor for combining temperatures would be expressed in terms of taking heat out of one kettle and putting it into the other. But the parallel between heat and temperature is still a difficult concept to grapple with [cite that reference].

On the other hand...

Temperature can be defined in a way that makes it proportional to the square of the speed of something, perhaps an average particle. This yields at least a natural nil concept, corresponding to a speed of nil for an average particle. The interesting observation here is that the zero number for the common measurement scales does not correspond to the natural nil of the physical quantity.

This definition also yields a basis for defining a combinational operator. Suppose $t_i=ks_i^2$. Then $t_1\#t_2=t_3$ would be defined to mean $s_1^2\#s_2^2=s_3^2$ — provided we could explain what it meant to square a speed, or to combine such things. (Tedious reminder: we are working with physical quantities, not numbers.)

- Counts, Frequencies, and Molarity

In some domains the measure consists of simply counting things, as in populations, committee or team sizes, and inventories. The unit is intrinsically defined as one thing, and we are simply counting how many things. The combinational operator is intrinsically defined as addition.

Combination must be done carefully, in terms of distinct things. Combining the sizes of two committees does not necessarily correspond to combining the committees, if some people are serving on both.

The unit notion has to be carefully distinguished from other properties. Bigger populations do not assure larger weight or volume. Walls can be measured in bricks, even if the bricks are of different sizes. Walls whose sizes in bricks are equal may have different weight, height, area, or volume. (Distinctness matters here, too. If two walls are connected — i.e., share bricks — then their combined size is not the same as the number of bricks in the two walls.)

Sometimes the things being counted are events, like births, or arrivals, or a cyclic phenomenon reaching a maximum or minimum. Frequencies or rates then correspond to counting the number of such things in some interval of time. This is the meaning of birth rates, arrival rates, and frequencies as expressed, for example, in cycles per second. Combining such things sometimes needs to be carefully distinguished from mixing together the underlying phenomena. Combining frequencies for measurement purposes does not correspond to composition of frequencies, as might arise from mixing tones.

Molarity is a form of concentration which counts the number of particles (molecules, atoms, ions) in a volume. Here count is serving as another variation of amount.

- Coordinate Systems

Coordinate systems provide two essential ingredients: an origin to establish the reference point, and a scale for measurement. In n-dimensional systems, there is also an “orientation” provided by the position of the axes. Is that significant here?

- A funny “unitless” example (mentioned by Bruce):

The electrical resistance of rectangular plates having the same composition and thickness is linearly proportional to length and inversely proportional to width. Hence the resistance is the same for all such plates having the same aspect ratio. In particular, the resistance is the same for all square plates. Therefore, the resistance of this material is specifiable as k ohms/square. Not “square something”, just “square”. Can we make a sensible dimensional analysis of that?

2.2 Units and Conversion

- Degrees of temperature and degrees of angle are different units which happen to have the same name. They measure unrelated domains.
- Degrees Fahrenheit and degrees Celsius are different units even though they measure the same domain. They are based on different elements of the domain.
- Degrees of circular measure and degrees of angular rotation are the same units even though they are measuring different domains. These domains are closely related.
- Are pounds of weight and pounds of mass the same units? If so, these related domains are not subsets of one another.
- Some units seem to be reserved for measuring specific things: hands for the height of horses, fathoms for water depths. Are furlongs used for much besides race courses?

2.3 Computational System Support

(Includes language and representations.)

- Syntactic matters:
  - Literal dimensional expressions.
- Language facilities for defining dimensions, compatibilities, units, etc.
- Tradeoffs between usability, rigor, consistency, etc. in language facilities.
- Internal representations, performance, etc.
- Concerns regarding precision, accuracy, context, etc.
- Which type coercions are legal?
- How should dimension, unit, precision and accuracy be inferred on input of data?
- How should units, precision and accuracy be specified on output? Should we say volts/ampere or ohms?

3 Basic Concepts

3.1 The Central Issue: Operational Consequences

For the purposes of computational support, the essential question is this: what happens when certain operations are applied to measurement terms? It breaks down into these parts:

- How are the operands recognized?
- Under what conditions is the operation considered to be an error? This relates to the notion of “compatibility”.
- When not an error, what are the quantities, dimension, and units of the result?
- What sorts of auxiliary operations are implicitly invoked, such as casting or units conversion?

[A little padding/transition needed here.]

3.2 Quantities Are Distinct From Their Expressions

The essential question is phrased in terms of computational operations, but that's not the place to start.

The appropriate perspective on all this work is rooted in a fundamental viewpoint: a physical quantity is a different thing from any of its expressions. Just as an athlete's performance is a different thing from a gold medal, so is the weight of a rock a different thing from an expression of the form "3 grams". The length of a certain stick is a certain concept, quite distinct from any such expressions as "6 feet", "72 inches", "2 yards", and so on.

The first obvious difference is that the length of the stick is just one thing, while we have a substantial number of different expressions which "represent" that length. It's just like the distinction between a person and all the different names and identifiers which might designate that person.

More important, we can imagine operations performed directly on such physical quantities. The combined lengths of two sticks is the same as the length of a stick which matches the first two sticks laid end to end. That operation can be done without any reference to numbers or arithmetic. One of the biggest sources of our difficulties with dimensioned data is a failure to appreciate this distinction. We confuse operations on physical quantities with arithmetic on numbers. One of the crucial things we have to do is to objectively assess the extent to which numeric operations do or don't mimic the actual behaviors of the underlying quantities, and to only use the arithmetic analogs when they do in fact correctly reflect the underlying behaviors.

In order to decide how the computational operations should behave, we should clearly understand how they correspond to the behaviors of the natural phenomena.

3.3 Some Terminology

We will call a thing which can be measured a *physical quantity*. Physical quantities can be categorized by *dimension*. A dimension describes the kind of physical quantity involved, such as time, length, mass, etc. In database and computing systems, we are interested in type-checking expressions representing physical quantities, and are thus concerned with the *type* of those expressions and their components.

Each dimension has an associated set of *units*. A particular physical quantity, such as an instance of Length, does not have any associated units. However, it does belong to a dimension, which defines a set of units which could be applied to it.

Some sources [3,4,5] use "physical quantity" to mean "dimension" as we have defined it here, and have no term corresponding to what we call "physical quantity." On the other hand, we do not limit physical quantities to things relevant to physics, so the notion can be extended to include other things such as money. We don't care what "physical quantity," "dimension" and "unit" mean; we merely specify here how things of those types will behave (of course, we intend for that behavior to closely model what is commonly meant by "physical quantity," "dimension" and "unit").

Algebraic relationships exist among some dimensions, such as

Force = Mass $\times$ Acceleration

Speed = Length/Time

Acceleration = Speed/Time = Length/Time^2

Current = Voltage/Resistance

Relationships among dimensions are not the same as something like

yards = feet/3,

which is a relationship between units, not dimensions.

A dimension may or may not have a name. Some dimensions are designated only by algebraic composition of other dimensions, such as Price/Weight or Weight/Length (which might correspond to such units as dollars/ounce or grams/cm).

Algebraic equations among dimensions can be rewritten with any of the dimensions as the dependent variable, defined in terms of the others. Hence, none of them is intrinsically more fundamental than the others. However, by convention [1, 3, 4, 5], certain dimensions are designated as the *reference* dimensions. The other dimensions are then called *derived*. The expression of a dimension as a multiplicative combination of reference dimensions, reduced to lowest terms (see [6]) is the *canonical form* of the dimension. This convention may affect the way dimensioned data is stored internally, and the way certain algorithms for dimensional analysis are executed, but has no bearing on the external semantics of dimensions.

For documentation purposes, we will ***[maybe?]*** find it useful to define a default alias syntax for naming any dimension, such that Length3.Time2'Mass names the dimension whose canonical form is Length ^{3}*Time ^{2}/Mass. Such notation will be used in this document to illustrate declarations on variables, and to mnemonically reflect the declared types in the names of the variables. Thus a variable named xLength3.Time2'Mass is restricted to values corresponding to that dimension. ***[Other notations, looking more like canonical forms or signatures are possible, but seem to run into problems when writing expressions. E.g., need to distinguish the name of a variable from an exponentiation operation on the variable.]***

Physical quantities can be further categorized into *scalar* and *vector* physical quantities. Scalar physical quantities are those whose representation needs only a single number (which could conceivably be a complex number ***[can't we dredge up an example?]***). Length, mass, and speed are all examples of scalar physical quantities. In contrast, vector physical quantities need more than one number to be represented. Velocity is an example of a vector physical quantity because it requires both a speed and a heading, such as 60 m.p.h. at 70 degrees. Position in n-dimensional space can also be thought of as a family of vector dimensions. Vector dimensions are discussed in Section 12. We consider derived dimensions to be multiplicative combinations of only scalar dimensions.

>>ToDo 2: Are any scalar dimensions represented with complex numbers?

3.4 Introduction to Units

In measurement, a "unit" denotes a particular mapping between physical quantities and numbers, for a given dimension. Thus, depending on the unit, my height might map into the numbers 72, 6, or 2, among others. At the same time, a unit denotes a distinguished physical quantity, namely the one which maps into the number 1 under the associated mapping. The "inch" mapping which maps my height into 72 maps a certain length into the number 1. That distinguished length is called an "inch". Under some other mapping, such as the one which maps my height into 6, it is a different length which maps into the number 1, namely the one we call a foot.

There is an essential parallel between physical quantities and arithmetic: any length is equivalent to some combination of 1-inch lengths or fractions thereof, just as any (non-negative) number is equivalent to a sum of 1's or fractions thereof. This is the basis of the units-based measurement paradigm, analyzed in greater depth in Section 11.

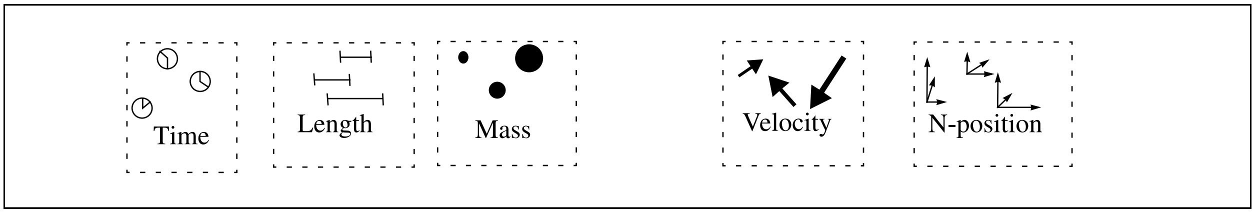
Units are distinct from their names. Just because "pounds" measure both weight and currency doesn't mean we are talking about one unit. The same can be said about "degrees" measuring both temperature and angles. In each case we have a pair of units which happen to have the same name. Conversely, although weights might be written as "1 pound", "2 pounds", "1 lb.", and "2 lbs.", we aren't using four different units. Those are all the same unit, which happens to have four different names or representations.

3.5 Dimensions as Types

[I seem to treat "type", "dimension", and "domain" as somewhat interchangeable, yet each seems to have its own connotation. Need to sort that out.]

To discuss rules about computations on dimensional terms, it makes sense to treat dimensions as types.

Dimensions have instances. All the lengths are instances of the type Length, all the masses are instances of the type Mass, and so on.



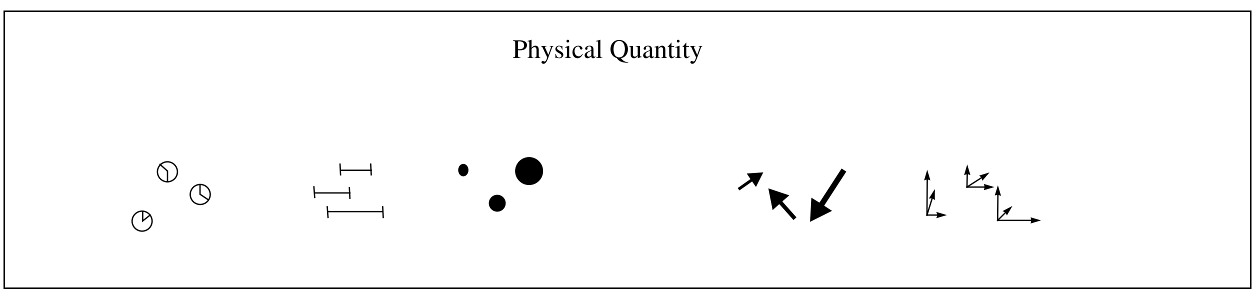
These types are somewhat different from traditional data types in that they don't have an associated data representation. It's as though we had an abstract Integer type whose values are simply the abstract integers, without restriction to a particular representation such as decimal, hexadecimal, binary, roman, etc. Specific representations for dimensions depend on the choice of units, which will be discussed later.

Spreadsheet programs treat dates like that. A column that's defined to contain dates isn't committed to any particular representation. That's a format specification, which can be different for different cells in the column, and can even differ for the same cell at different times. Despite these format differences, the spreadsheet program always knows that this column contains dates. If no format is specified, the spreadsheet program can always fall back on a default format.

Like types, dimensions will be used to constrain the values of variables, and also of the arguments and results of functions and procedures. Thus a given variable (or function argument or result) might be declared to be of type Length or Mass.

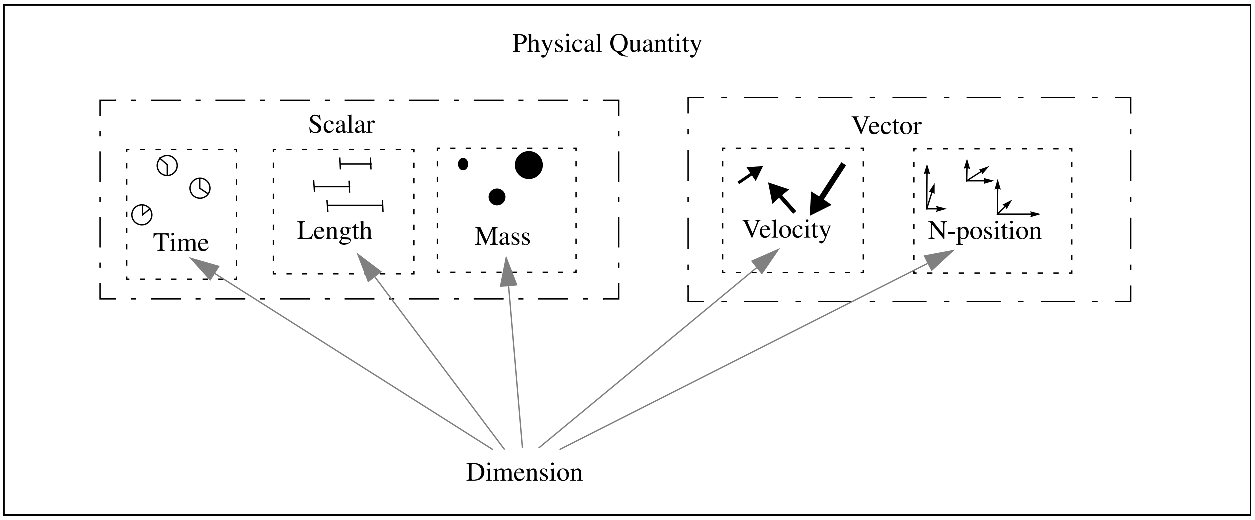
Similarly, the allowable operands of operations will be constrained by such types. Addition will be valid between two Length terms or two Mass terms, but not between a Length term and a Mass term.

The union of the dimensional types is also a type, comprising a supertype of each dimensional type. Its instances are all of the physical quantities; we call this type Physical Quantity (or PQ).



The type Physical Quantity is covered by its subtypes—every physical quantity belongs to some dimension. The subtypes of Physical Quantity are instances of the type Dimension. ***[Name conflict: operation and type.]*** For example, Length and Mass are subtypes of Physical Quantity and instances of Dimension. The type Physical Quantity is characterized by an operation we might call Dimension, which returns the dimension of any physical quantity. Thus Dimension(x) might return something like Length or Mass as its result.

>>ToDo 3: Sort out name conflict: Dimension as operation and as type.



This type structure is analogous to employees and jobs. The type Employee might have subtypes Engineer and Programmer. The type Job might have Engineer and Programmer as instances. Thus Engineer would be both a subtype of Employee and an instance of Job.

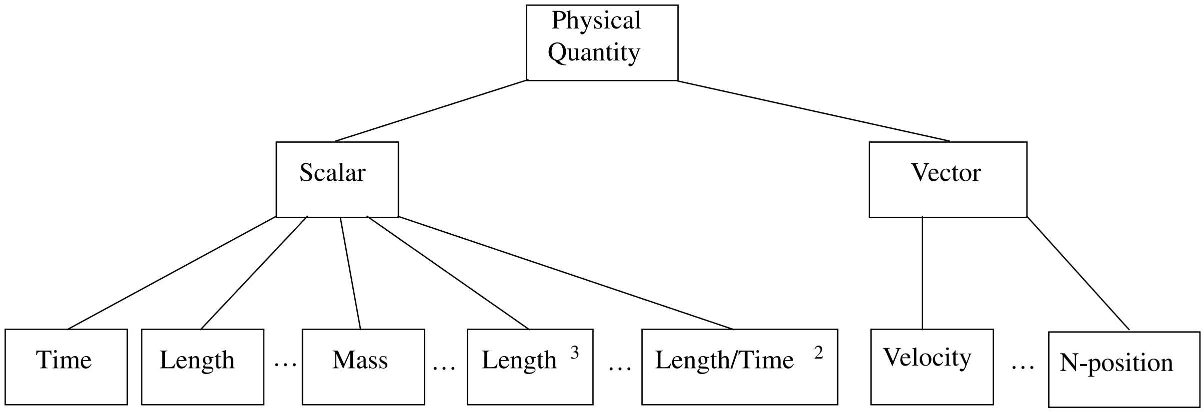
The subtypes of Physical Quantity are not necessarily disjoint. An exception to this is quasi-dimensions , introduced in Section 9.

Base types such as Length and Mass are explicitly declared. There is also an implicitly defined infinite set of types, consisting of the reduced multiplicative combinations of the base types. Thus Length^2 and Length/Time are implicitly defined types, being respectively the same types as Length^3/Length and Length*Time/Time^2. These are somewhat analogous to generated (parameterized) types such as Array(T) or Set(T), which implicitly define a family of types, one for each type T. ***[Can we provide a reference for that?]***

The set of dimensional types is thus infinite. In a real system or language, the base types will be explicitly articulated, while most of the implicit types will not. Some implicit types will be articulated in order to define additional specifications, such as aliases [Section 4], specializations [Section 8] or unique units. Length/Time might be explicitly listed, for example, in order to define Speed as an alias or specialization and mph as an associated unit.

It's common practice to display such a set of types as a graph. Of course, it's only feasible to exhibit the explicitly articulated ones, and not the entire infinite set of implicit types. At present, the only relation represented by the edges between type nodes in the graph is subtyping. Other relations will be introduced later.

Including derived and vector dimensions yields the expanded type hierarchy:



3.6 Precision and Accuracy

Our knowledge of physical quantities often has finite precision and accuracy. For example, a manufacturer of low-cost batteries may specify their output voltage as "1.5 volts, ±0.1 volts," meaning that the design average of all their batteries is exactly 1.5 volts (infinite accuracy), but that any given battery is allowed to lie 0.1 volts either side of that (finite precision). The morning after a burglary, a factory inventory system may know the contents of a bin of bolts to a precision of 1 (infinite precision, assuming no fractional bolts), but its value may be off by 100% (more formally, its accuracy is something like +0%, -100%).

Precision and accuracy affect the manipulation of data about physical quantities in the following ways:

- Arithmetic operations yield a new quantity, with different precision and accuracy.
- Precision and accuracy need to be carried with physical quantities when they are stored or assigned to variables. Unfortunately, we know of no standard method for representing precision and accuracy. Logically both are distributions, and require potentially very many numbers to describe them. In practice [2], people describe precision in number of significant digits or the width of the interval between readings. For accuracy they use 0-3 parameters, but they differ in number of parameters and their semantics (mean, stdev; mean ±n; mean+n, mean-n; etc.). System designers will have to deal with this variability.
- A new kind of equality is introduced, which takes precision and accuracy into account (it may say, for example, that two quantities are equal if they are equal within the precision of the most precise).

Note that physical quantities themselves are not uncertain—it is our knowledge of them that is uncertain. Note also that this uncertainty is present even *before* we try to represent the quantity as a PQE in any computer or on any piece of paper. Further degradation occurs when we try to do those things; it is discussed in Section 34.6.

See ***[Ref which Dan will find]*** for a more complete discussion of precision and accuracy.

In general, any finite representation of a number actually specifies a range of values. According to common rounding rules, the decimal representation "3" means "somewhere in [2.5, 3.5)." The more precise the representation of a number is, the narrower is the band of numbers it specifies. Low precision thus corresponds to wide bands of numbers. The numbers in measurements have less-than-infinite precision for two reasons. First, they have finite representations (e.g., 64 bits). Second, they may represent quantities that we know with only finite precision. For example, an architect may specify lengths only to the nearest 1/16." If a drawing says that a wall should be made 3-5/8" thick, it would be an overspecification error to record the thickness as 3.6249999999." Such a specification would give rise to walls that were a satisfactory thickness, but which cost too much. Similarly, a geologist may use dating techniques which are reliable only to the nearest 1000 years. It would be scientifically dishonest to report that a rock was formed on "April 3, 3,000,025 B.C."

Note that finite-precision numbers need not come from measuring anything. The converse is not true: measurements always yield finite-precision numbers.

Precision is different from accuracy. We can say that π is 3.00000, for example. That specifies the number to a precision of about 0.00001%, but its accuracy is only about 4.5%. Loss of accuracy can come internally from cumulative arithmetic operations at finite precision, and externally from measurement bias and errors of all sorts.

Although accuracy and precision are independent notions, a representation of a quantity can never be more accurate than it is precise. Note also that accuracy is a property of measurements, while precision is more a property of the *representation* of measurements (or of specifications).

3.7 Context

Knowledge about physical quantities exists in context: when an observation was made, by whom, using what method and equipment, under what ambient conditions, etc. Which context is relevant is application-dependent, but it may include the date/time, something to identify the measurement apparatus and its level of calibration, the name of the operator, operator's comments, ambient temperature, etc. Note that this information is about the *observation*, not about the quantity observed.

Context also affects unobserved physical quantities: a specification for the length of a steel bar *at a given temperature,* the expected height of a tree *on a given date,* etc.

As with precision and accuracy, context affects the manipulation of data about physical quantities:

- Arithmetic operations yield a new quantity, with different context. It is application-dependent what the relevant items of context are, and how they are generated. For example, the sum of lengths L1 and L2 might include in its context an ambient temperature which is the average of the ambient temperatures of L1 and L2. The sum's time stamp might be the time at which the sum was computed.
- Logically, the assignment operator copies all context information. In practice, it is application-dependent which components are copied.
- As with precision and accuracy, a new form of equality is introduced, which takes the relevant context into account.[1] Again, it is application-dependent which context is relevant.

>>ToDo 4: Is there any further discussion of context anywhere?

1. Logically, there is one for each subset of the components of context, which takes that subset into account, or there is one form, parameterized by the subset which it takes into account. We speculate that in practice the number of different subsets actually used will be manageable

II INTER-DIMENSION RELATIONSHIPS

There is only one relationship among dimensions in traditional dimensional analysis. Any dimension can be expressed as a multiplicative combination of base dimensions.

We solve many of the problems described earlier by introducing new kinds of relationships among dimensions.

***>>ToDo 5:* Need to complete all aspects of type relationships and graphs.**

Notes...

How we know whether something is of a certain type should probably be pursued in detail in the section on language stuff, but we need to say something about that here.

Implications for the type graph, canonical forms, compatibility, etc. really need to be worked out.

Need to work out the conditions under which dimensions comprise a "units family", sharing a common set of units. When can additional units be defined for a particular dimension in the family? Also need to stop relying on units to infer the dimension of a quantity.

4 Independent (Orthogonal?) Dimensions: Dimensional Analysis

In traditional dimensional analysis, there is a finite set of base dimensions, such that all other dimensions can be defined as multiplicative combinations (meaning multiplication or division) of the base dimensions. The base dimensions are orthogonal, in the sense that no base dimension can be defined as a multiplicative combination of other base dimensions.

The choice of base dimensions is somewhat arbitrary, and we don't endorse any particular base set. Base dimensions used in this document are for illustrative purposes. Any real computational support will embody a particular set of base dimensions, probably with the ability to define new ones as well. ***[But mention/reference standards.]***

Any multiplicative combination of base dimensions in reduced form is a distinct dimension. ***[Is that in dimensional analysis, or our extension?]*** Length^2 and Length/Time are dimensions, being respectively the same dimensions as Length^3/Length and Length*Time/Time^2. Any dimension can be expressed in *basic canonical form* as a multiplicative combination of all base dimensions, i.e., a product of all the base dimensions raised to powers which might be zero, positive, or negative. Acceleration has the basic canonical form

Time^{-2} * Length^1 * Mass^0 * ...

If a fixed ordering of the base dimensions is established, then the *basic signature* of a dimension can be expressed as a sequence of exponents of the base dimensions. ***(If there's some other common term for this, we can do a global replacement.)*** If Time and Length are the first dimensions in this sequence, then the basic signature of Acceleration would be

-2 1 0 0 0 0 ...

Signatures are expressed more compactly in this document via the following conventions:

- Trailing zeros are elided.
- 0^n denotes a string of n zeros.
- $\overline{n}$ denotes -n.

Thus $1\ \overline{2}0^5 3$ denotes the signature 1 -2 0 0 0 0 0 3 0 0 0. The compact signature of Acceleration is $\overline{2}1$.

Such signatures are simply an expository notation for expressing a sequence of powers of base dimensions, and does not imply anything about representation of dimensions in an implementation or programming language.

The names given to certain derived dimensions, such as “Area” or “Speed”, serve as *aliases* for these dimensions. (A documentation convention for aliasing any dimension is described in Section 3.3.)In algebraic operations, aliases and their canonical forms can be freely interchanged. Aliases are unique, in the sense that each alias names exactly one dimension. (The situation will be somewhat different for specialized dimensions [Section 8].) Dimensional analysis generally proceeds by replacing all aliases with their unique canonical forms, and then performing algebraic reductions as though the dimensions were variables.

In terms of signatures, the main computational rules of dimensional analysis are:

- The signature of a product of dimensions is the vector sum of the signatures of the terms.
- The signature of a quotient of dimensions is the vector difference of the signatures of the terms.
- Dimensions are compatible for purposes of addition, comparison, or assignment if their signatures are the same.

[We may want to include or reference more detail, such as treatment of constants and formation of the initial expression for dimensional analysis.]

5 Subtypes

[Add some introduction as to what subtype means, and how it differs from other type relations.]

5.1 Unsigned and Signed Domains

The domains of unsigned (positive) and signed lengths have the following relationship:

- The unsigned (positive) lengths constitute a subset of the signed lengths.
- All operations applicable to signed lengths are applicable to the unsigned lengths, except that the behavior of subtraction is redefined for unsigned lengths. x ~y is valid for any pair of signed lengths. For unsigned lengths, if x<y then x ~y yields an error; otherwise x ~y is the same as for signed lengths.

This relationship fits at least one of the definitions of “subtype”, and we can consider unsigned lengths to be a subtype of the signed lengths.

With these types distinguished, it is clear that x ←(3 feet − 4 feet) yields a valid result if x is a signed length, but is an error if x is an unsigned length. (The subtraction could be valid, but the assignment isn’t.)

Any domain which has total ordering and nil comes in such a signed and unsigned pair. ***[That’s not quite accurate. Circular angles are closed under subtraction; negatives are not disjoint from positives. So, refine this observation.]***

>>ToDo 6: Complete the stuff on subtypes, unsigned/signed domains.

Questions:

- Other sorts of subtypes?
- What impact on canonical form? Declarations? Literals? How do we know when something is of which type?

5.2 Others?

>>ToDo 7: Are there other examples of subtypes?

6 Point and Interval Types

[Note that this concept applies to scalar dimensions, although vector dimensions might have point components.]

Certain anomalies can be resolved by distinguishing between the point and interval forms of certain dimensions. Thus 5 °C and 41 °F denote the same temperature point (how hot something is), while 5 °C and 9 °F denote the same temperature interval (how much hotter one thing is than another). Similarly, a heading of 90° is simply a single directional point — that's where a ship is aimed — while an angle of 90° denotes an interval, i.e., how much a ship might turn. The fundamental analogy is with points on a line and the distances between pairs of points.

The intuition is hard to pin down precisely, but a point quantity has the sense of being a single thing, while an interval quantity expresses a relationship between two things. For some dimensions, there is no distinction. Thus the difference between two weights is still a weight, and the difference between two lengths is still a length.

The units-based paradigm that we've been discussing applies to interval dimensions. Let's say that any domain with units-based measurement is an *interval* domain. ***[May need to reconsider that.]***

>>ToDo 8: Refine the definition of interval domain.

A *point* domain has a total function Diff which maps a pair of points onto an element of an associated interval domain: $\mathrm{Diff}(p_1,p_2)=i$. Equality in the point domain is defined in terms of the null element of the interval domain:

- $\mathrm{Diff}(p_1,p_2)=\varnothing$ if and only if $p_1=p_2$.

This difference is undirected (unsigned) i.e.,

- $\mathrm{Diff}(p_1,p_2)=\mathrm{Diff}(p_2,p_1)$.

We can have $\mathrm{Diff}(p_1,p_2)=\mathrm{Diff}(p_1,p_3)$ without $p_2=p_3$ if, for example, p_1 is between p_2 and p_3.

Directed differences can be introduced if the associated interval domain is signed, so that

- $\mathrm{Diff}(p_1,p_2) = -\mathrm{Diff}(p_2,p_1)$
- $\mathrm{Diff}(p_1,p_2) \neq \mathrm{Diff}(p_2,p_1)$ for $p_1 \neq p_2$.

Given such a function, a useful "measurement" of points can be defined relative to some fixed origin point. Let M be a function which returns some measurement of an interval. The measurement of a point p relative to an origin point o can be defined as

$$M_o(p)=M(\mathrm{Diff}(p,o)).$$

Such a measure is clearly dependent on the choice of origin o. ***[What more needs to be said about directed and undirected stuff?] [Generalize: the result of a measurement is dependent on scale (unit interval size) and origin.]***

>>ToDo 9: Generalize the units concept to include origin?

An interval domain is in some sense a degenerate point domain which maps into itself and always takes the natural zero as the measurement origin. Thus we can postulate (or define?) that every point domain has an associated interval domain; an interval domain has itself as the associated interval domain. Another way of saying this is that the result of a difference operation is in the associated interval domain. Interval domains are closed under difference, while point domains are not.

Measurement in a point domain is defined indirectly via measurement in its associated interval domain. Since the measurements are so expressed, it follows that the units for the point domain are “inherited” from the interval domain.

[More examples!]

[It doesn’t seem possible, based on these definitions, for a point domain to have any unique units of its own. However, it might have unique instance representations of its own, such as the compass points. Interesting observation: N, E, NNE, etc. are not “measurements” but representations of specific quantities. Is that also true of letter grades? Stop and think: is this different from measurement, or not?]

>>ToDo 10: Do point dimensions have units? Are they really measurements?

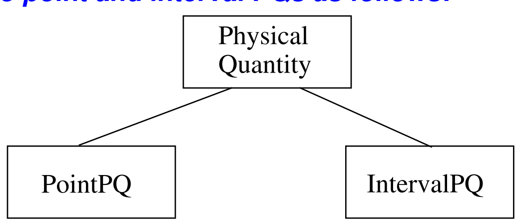
[The underlying line and point metaphor leads naturally to thinking about location, but that’s only in one-dimensional space. Say something about how this extends to other notions of location, which would get us involved with vector dimensions.]

[Discuss the nature of the relationship between point and interval dimensions. It’s hard to justify a subtype relationship among point and interval domains, even though they share such things as units. Their populations really are different things.]

>>ToDo 11: Clarify the nature of the relationship between point and interval dimensions.

Stephanie’s question: If point “dimensions” are not PQs in the sense as defined in Measurement Fundamentals, what are they? Are they siblings of PQ?

Proposal: split PQ into point and interval PQs as follows:



6.1 (Old Material)

>>ToDo 12: Check relevance of this other old material.

One category of physical quantity is closely related to the kinds of quantity we discuss in this paper—counts of some unit along a dimension—but is not itself such a quantity. Dates, spatial locations, computer memory addresses, and headings are examples of this category: the location of Palo Alto, California isn’t a length, and it doesn’t make sense to ask how “big” it is. We call dimensions of this sort *point* dimensions. The distinction between point dimensions an other dimensions is confusing because headings are often measured in degrees clockwise from North, memory addresses are often described by a count of bytes from location 0, dates are often described by time intervals from an (implied) origin, and points in space are often described by their distance from an (implied) origin. But to say that Palo Alto is “at” 37° 26’ 31” North, 122° 08’ 31” West is no different from saying that it’s 44 miles south of San Francisco—both describe a related distance, but do not “measure” the location itself. This characterization is consistent with [1], which treats points and intervals as distinct dimensions.

Elements of point dimensions are sometimes referred to by names, rather than described by a related magnitude. Points in space are often named by a city or geographical feature; dates are sometimes known by the name of the current monarch or some important event; and headings are sometimes named with a combination of the names of the compass points.

But when elements of a point dimension are described by the measure of some physical quantity, the quantity always belongs to a related dimension. Locations in space are described by intervals in space (i.e., lengths), memory addresses are described by numbers of bytes, dates are described by intervals in time, headings by plane angles, etc. We call this related dimension an *interval* dimension. The touchstone for whether a dimension is of the "point" or "interval" type seems to be whether it is closed under subtraction. [1] If X and Y are the same kind of thing (belong to the same dimension), but X-Y is a different kind of thing, then X and Y belong to a point dimension and X-Y belongs to the related interval dimension.

There can be more than one interval dimension related to a given point dimension. For example, points on the earth's surface are sometimes described by their distance from an origin along a path (usually a straight line), and sometimes by their angular offset from the Equator and the International Date Line. ***Is the converse also true?***

Special compatibility rules apply to point dimensions and their related interval dimensions. Let **P** be a member of the point dimension, **I** be a member of the related interval dimension, and **U** be a number. Then:

- **P** + **P** is undefined
- **P** + **I** → **P**
- **I** + **I** → **I**
- **P** - **P** → **I**
- **P** - **I** → **P**
- **I** - **P** is undefined
- **I** - **I** → **I**
- **U** ± **I** is undefined
- **U** × **P** is undefined
- **U** × **I** → **I**

A system which supports points and intervals needs to know which dimensions to treat this way. At a very high level of support, it could allow users to make their own specifications.

Some anomalies are worth pointing out.

- Most point quantities are expressed in the dimension of their related interval dimension: locations as Distance or Angle, headings as Angle, temperature points (discussed below) as Temperature interval, etc. Dates are an exception: although the expression "January 6, 1914" contains a number of years since an implied origin, it is not generally taken to mean 699,095 days. It is arguable whether the expression "5 million years B.C." is a negative or positive quantity.
- The interval dimensions Length, Time and Angle all measure a relation between two quantities, rather than a property of one quantity. Length measures an interval between two points; Time measures a duration between two points, and Angle measures a relation between two lines (or lengths). Any description of the property measured by these dimensions will name two points. Temperature is different. It is an interval dimension, but it measures a property of single quantities.

---

1. Actually, whether it is closed under the Δ operator mentioned in Section 11.1.7.

- Physics offers no natural origin for measuring headings, dates or spatial locations (North, the birth of Christ, and Greenwich all come from modern, human considerations). Temperature, on the other hand, has a physical zero (a "nil" value). We thus have several temperature scales, one with zero measurement corresponding to nil, and the others with zero measurement corresponding to human-sensible values.
- We are not sure how to treat the temperature dimension. On the one hand, temperature points and temperature intervals are different things (careful use even gives them different names: "degrees Celsius" and "Celsius degrees"). On the other hand, while Length measures a relation between two things, and duration measures a relation between two things, temperature measures a property of one thing. While there is no origin of space or beginning of time, there is an absolute zero temperature. For temperature, therefore, there is a choice of origin which gives an isomorphism between points and intervals. It is therefore at least unambiguous, if not meaningful, to compute the sum of two temperature points, to calculate half the temperature in San Francisco today, etc. One can construct arguments both for and against having a system do this without complaint. This area needs further work.

6.2 Temperature; Zero-Origin Units

>>ToDo 13: Does this treatment of temperature make sense? Consistent with our other treatments?

Temperature behaves somewhat like a point domain, yet it has a natural zero. We can devise an interesting analogy with weights. If many things in our experience weigh 100 pounds or more, we might have another unit called "phounds", expressing the number of pounds above or below 100. Something weightless is -100 phounds, while a big sack of potatoes might weigh 0 phounds.

Two things weighing 1 phound each weigh 102 phounds together. If x=1 phound, then 2x=102 phounds. Sounds like temperature.

[But how do differences behave?]

[Maybe gauge pressure, being relative to atmospheric pressure, is another analogous case.]

We could generalize the notion of units and say that a unit expresses both a scale and an origin, with the origin often but not always coinciding with a natural zero. Arithmetic is only valid with units based on a natural zero. Expressions with other units must first be mapped to a zero-origin unit, if possible. This formulation accounts for the polynomial form for temperature conversions, since both scale and origin might have to be transformed.

Conclusion: temperature is an interval domain, but its popular units are not zero-origin.

7 Siblings

7.1 The Family of Angles

[To be completed. The whole story on angles may involve some combination of siblings and specializations. On the other hand, the whole concept of siblings might break down on more careful examination.]

>>ToDo 14: What does the sibling relationship really mean?

>>ToDo 15: Sort out the structure of the family of angle dimensions.

[Also at Section 7.1.5.]

Angle dimensions may turn out to have a rather complex structure. One thing they all have in common is units of degrees. Most also have units of radians, but that probably does not apply to compass headings. On the other hand, compass points probably don't apply to anything but compass headings.

How do compass headings fit in if they are a point dimension while all the others are interval dimensions?

How do these relate to each other? What sorts of inheritance go on? [There is some discussion at Section 7.1.5.]

Do the time dimensions behave similarly, or do they introduce something new?

7.1.1 Rotational Angles

Rotational angles measure how much something has been rotated. Turning something full circle is different from not turning it at all, and doing that twice yields something different, and larger, again. One good example measures how many times a screw is turned.

This is a very nicely behaved domain, exactly isomorphic with lengths, even to the extent of having signed and unsigned variants.

To begin a running example, in rotational angles we have:

- $360° > 180° > 0°$
- $280° + 280° = 560°$

A curious observation: the notion of a circle has little significance in this domain. There's nothing very special about 360°. A rotation of 360° sort of gets you back where you started, but not exactly. However, since it will be useful later, we will use Θ to denote a rotational angle corresponding to one full revolution, while η will denote the nil angle (0°). Other angles will often be denoted by their measurement in degrees.

7.1.2 Modulo Domains: Circular Angles

This is the familiar circular measure in which all angles lie between 0° and 360° (η and Θ).

Circular angles constitute a subtype of the rotational angles, containing just the instances $\eta \le x < \Theta$. The significance of that half-open interval must be clearly understood. The rotational angle Θ is the least upper bound for all the circular angles, but it is not itself a circular angle. For circular angles, "360°" is just a synonym for "0°", and is very different from the rotational angle denoted by "360°".

Circular angles inherit units, equality, and order from rotational angles, but addition and subtraction have to be redefined.

The combining operation is slightly different for circular angles, since $359° \# 1° = 0°$. It's easiest (and perhaps unavoidable) to define the behavior of # for circular angles in terms of its behavior for rotational angles. Note that we want to talk about angles, not numbers. We want to justify the adoption of modulo arithmetic, not assume it. Inevitably, the description will sound much the same.

To combine two circular angles, first combine them with the rotational # operator (it is applicable, since the angles exist in both domains). If the result is larger than Θ, subtract Θ (in the rotational domain). The result will be in the range $\eta \le x < \Theta$, hence is the desired circular angle.

Exercise: verify that circular # is associative and commutative, without resorting to modulo arithmetic.

It would seem trivial to claim now that we redefine addition as modulo addition to capture the behavior of circular #. If $x_1 = k_1 u$ and $x_2 = k_2 u$ for some unit angle u, and $\Theta = k_3 u$, then

$$x_1 \# x_2 = (k_1 + k_u) \backslash k_3 \, u.$$

That is, the measure of the combination of two angles is the sum of their measures modulo the measure of a full circle.

Unfortunately, we haven't quite justified the notation k_1u.

What can we say about multiplication, i.e., rx for some real number r and angle x? It's tricky. The first step, unique definition of kx for any non-negative integer k, seems to be all right. But then we notice that we might have $k_1x=k_2x$ with $k_1\neq k_2$, if $k_1-k_2=n\Theta$ for some integer n. Thus, for a given pair of angles x_1 and x_2, there can be many pairs of integers k_{i1} and k_{i2} such that $k_{i1}x_1=k_{i2}x_2$, with the ratios k_{i2}/k_{i1} being all distinct. Hence the rational number r in $x_1=rx_2$ is not uniquely determined. Hence we don't have a foundation for units-based measurement, unless we find a paradigm for selecting a unique r.

For example, if $x_1=60°$ and $x_2=90°$, we have $3x_1=2x_2$, $9x_1=2x_2$, $3x_1=6x_2$, and so on. If we don't find some way to prevent it, we could infer that $x_1=(2/3)x_2=(2/9)x_2=(6/3)x_2$, etc. What exactly makes us believe that 2/3 is the correct ratio? It's neither the minimum nor the maximum ratio. But it does seem to be the one with the smallest numerator and the smallest denominator. Is that a provably consistent and correct algorithm?

Here's the outline of a proposed definition, with key steps remaining unjustified:

- For a given pair of circular angles x_1, x_2 with $x_2>\eta$, let $\{k_{i1}\}$ and $\{k_{i2}\}$ be sets of positive integers such that $k_{i1}x_1=k_{i2}x_2$. If either set is empty, quit.
- Let k_{*1} and k_{*2} be the minimum values in each set.

  [It's evident that these minima exist, isn't it?]
- *Define* k_{*2}/k_{*1} to be the *ratio* of x_1 to x_2. If $r=k_{*2}/k_{*1}$, we can write $x_1=rx_2$.

  Pitfalls:
  - How do we know that $k_{*1}x_1=k_{*2}x_2$? k_{*1} and k_{*2} might have come from different equations.
  - Can we satisfy ourselves that k_{*2}/k_{*1} is the intuitively correct ratio?

Eureka! There's a simple solution to that problem! The previous exercise has been an extremely valuable demonstration of the care required to avoid false assumptions about arithmetic in measurement domains. However, it takes the wrong approach for our present purposes. [So, let's keep that other material somewhere as a demonstration.]

An elegant solution is based on a trivially simple intuition: we say that $x_1=rx_2$ as circular angles if and only if $x_1=rx_2$ as rotational angles, for any real number r. That's all it takes.

Consequently, rx is a defined and unique circular angle for any circular angle x and any real number r.

(Observe that we depend in several places on the circular angles being an actual subset of the rotational angles. That's what allows us to compute with circular angles as though they were rotational angles whenever we need to.)

[Still need to justify the fact that we recognize 400° as denoting the circular angle also denoted by 40°.]

Subtraction also turns out to have interesting properties. Circular angles are closed under subtraction. That is, for any x and y, there exists a z such that x#z=y, hence x~y=z. This means that the negative circular angles are the same as the positive circular angles, and there is no need to distinguish between unsigned and signed circular angles.

Exercise: demonstrate uniqueness of x~y. Don't get confused by the observation that 60°#90°=150° and 60°#450°=150°, suggesting that 150°~60° is non-unique. Once again, 90° and 450° are not different circular angles, but different notations for the same angle. Hence the difference is in fact unique.

One last curiosity: as with modulo addition, the monotonic nature of combination is lost. That is, we can have $x\#y<x$, even if $x>\eta$ and $y>\eta$. In fact, we could have $k_1x<k_2x$ though $k_1>k_2$ (e.g., 2*200<1*200). This demonstrates that ordering has to be postulated independently, and can't be inferred from the combining operation.

To continue the example, in circular angles we have:

- $180° > 0° = 360°$
- $280° + 280° = 200°$

7.1.3 Interior Angles

Here's a strange though familiar domain, in which no angle seems to be larger than 180°.

Did you ever see two fences meet at an angle of 270°? Did you ever see a 270° angle bracket? Would you ever describe two streets as intersecting at 270°? These are all generally considered the same as 90°.

Would you ever say that two fences, or two streets, intersected at 180°? Is there such a thing as a 180° angle bracket?

Do you doubt that these angles are measured, or added, or compared? Try discussing this at your neighborhood hardware store.

A similar notion of angle seems to underly the assertion that the base angles of an isosceles triangle are equal.

As long as everything is between 0° and 180°, things behave quite nicely. But once we cross 180°, look out! We have $180°+x=180°-x$, even with $x\neq0$. We have $170°+20°=170°$. As we accumulate 1° angles the results get larger and larger — until we get to 180°, after which the angles get smaller and smaller while we keep accumulating in the same direction.

In effect, the combinatorial operation # includes flipping the angle around if it exceeds 180°.

>>ToDo 16: Complete the investigation of interior angles.

Exercise: which axioms are failing here? Unique coefficients? Associativity? Unique nil? Distinguish nil from an annihilator.

[Add uniqueness of nil as an axiom? Without unique nil we could have problems with negatives.]

(Should check whether # is associative in this domain. Can't tell for sure. Should observe that associativity is verifiable in circular angles and other modulo domains — if there are any.)

7.1.4 Compass Headings

Notes:

We could treat compass headings as being isomorphic to circular angles, with the addition of letter notation for certain angles. But one could also make the case for compass headings being a point domain, with arithmetic and order being inapplicable. This seems to be the natural semantics of compass headings.

Should we adopt some other name for these, such as bearings or direction?

>>ToDo 17: Complete the investigation of compass headings.

7.1.5 Summary of Angles

As before, the reason for distinguishing such types is to declare differences in intended behaviors. The ambivalent intent of certain operations can be resolved by defining distinct types, each corresponding to a different behavior of the operation. Thus op(x) behaves one way if x is of one type, and another way if another type.

Rotational	Circular	Interior	Heading
360°>180°>0°	180°>0°=360°	360°=180°=0°	360°=0°≠ 180° x>y: error
280°+280°=560°	280°+280°=200°	280°+280°=160°	280°+280°: error

An old message to be re-emphasized one more time again: don't let the numbers fool you! In circular angles, 360° is not bigger than 359°. 360° and 0° are just different names for the same angle, which is less than all other angles in circular measure. Circular measure is a *bounded* domain. All the circular angles are very nicely totally ordered, even the ones very close to that upper bound. There is a least upper bound, but it's not in the domain.

In circular angles, 560° is just another name for the angle called 200°. In fence angles, 280° is just another name for the angle called 100°, while 560° and 200° are both synonymous with 160°.

Speculation regarding type structure: Circular and interior angles could be subtypes of Rotational. Heading probably can't, since it doesn't inherit certain operations. Anyhow, Heading is a point type while the others are interval types. If Heading and Rotational are siblings, it would be nice to have a common supertype from which to inherit things like units. ***[Problem?]***

[We need more wrap-up here. What are the general principles? To what other examples would they apply?]

7.2 Other Examples

>>ToDo 18: Are there other examples of sibling dimensions?

8 Specialization

>>ToDo 19: Do we have consensus on this treatment of specialization?

8.1 Purpose

Work and torque both have the same canonical representation—Force × Length—but they aren't the same thing. Work is a force applied along its direction of travel, while torque is a force applied at some distance from a center of rotation, such as a force applied to the handle of a wrench. Thus work and torque are partially equivalent, in the sense that they have the same canonical representation and hence appear equivalent under dimensional analysis, but they are not simply substitutable for one another. In a sense work and torque are different "specializations" of the dimension Force × Length.

>>ToDo 20: Is torque a vector?

[One can argue that Torque is a vector, whereas Work is a scalar, and that distinguishes them. This discussion still applies to the magnitude of the torque vector.]

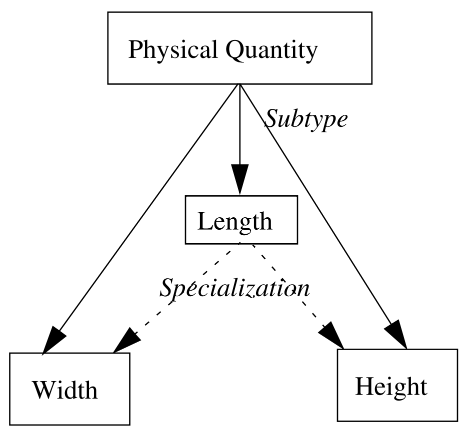
Other examples: the productivity of a lumber mill or licorice factory might be expressed in feet per hour, but this is not the same thing as Speed. Fuel consumption in gallons per mile is Volume/Length, but is not the same thing as Area. Paint coverage in gallons/square foot is Volume/Area, but is not the same thing as Length. In some contexts it is convenient to treat Length, Width, and Height as different dimensions. In some contexts, Height^2 is not compatible with Area.

Similar issues arise with "dimensionless" dimensions. For purposes of dimensional analysis, Angle is equivalent to Length/Length [1]. Certain concentrations might be expressed as Weight/Weight, or as Volume/Volume. The accuracy of a timepiece might be expressed in seconds/year, i.e., Time/Time. All of these have a dimensionless canonical form, and are thus equivalent under the textbook treatment of dimensions.

However, one may wish for a system supporting dimensions to treat them as distinct. In the same way that a good CAD system can raise an error if two different signals are applied to the same wire (even though both are of type "signal"), a system supporting dimensions can raise an error if a user tries to assign a variable of type work to a variable of type torque, or compare a lumber mill's output to a runner's speed, or add a clock's error to the concentration of a sugar solution.

8.2 The Specialization Relationship

S*pecialized dimensions* enable the kinds of distinction mentioned above. A specialized dimension is defined in terms of an existing dimension, which is then the *parent* of the specialized dimension. Specializations of the same parent are *siblings*. ***[Watch our for conflicting usage in Section 7.]*** For example:



[Might later change the example in the figure. Could show a real subtype relationship, such as positive (unsigned) lengths under signed lengths. The contrast between subtype and specialization could then be made clearer. Incidentally, this also suggests more motivations for unsigned lengths as a distinct type: might not want things like area, volume, speed, etc., expressed in terms of signed lengths.]

Specialized dimensions are themselves dimensions, and may be further specialized. However specializations of a parent dimension **D** are not subtypes of **D**. Rather, they behave like what the relational algebra calls "domains" [12, 14]. Briefly, a domain defined over **D** inherits all the properties of **D**, but doesn't obey the "is-a **D**" relation, so its elements aren't automatically promoted to elements of **D**. This behavior is what makes Width and Height incommensurate in the example above (or Energy vs. Torque, Length/Length vs. Time/Time, etc.).

The essential difference between specialization and subtyping can be described in terms of inheritance. The intent is to prevent a width from being added to a height. However, if Width and Height were subtypes of Length, then the two terms, being also instances of Length, could still be added as lengths. Making Width and Height specializations rather than subtypes prevents this.

A specialized dimension might have multiple parents. Area might be defined as a specialization of Longness*Width and also of Longness*Height. This introduces some problems, discussed in Section 8.4. Until then, we assume each specialization has a single parent.

[“Length” has been adopted as the standard term for the base dimension, where we might sometimes find “distance” more convenient. We therefore coin “longness” as the third counterpart to width and height. Thus, longness, width, and height are three sorts of lengths.]

A specialized dimension inherits the units and unit conversions of its parent. Additional units can be defined especially for the specialization, such as mph for Speed or “hands” for Height (often used for the height of horses).

The specialization relationship is also significant in managing compatibility (below).

8.3 Compatibility Levels

Specialized dimensions prevent users from making certain mistakes, such as adding a work value to a torque value. This reflects a protective philosophy of language design, in contrast to a permissive philosophy which assumes the user knows what he is doing and shouldn't be impeded by the system. Designers of any real computational support system will have to decide where they stand on the protective vs. permissive spectrum.

There is a spectrum of possible compatibility enforcement strategies.

8.3.1 Strictest Enforcement

The strictest level of enforcement is simple and consistent, but quite inconvenient. Specialized dimensions are considered just as distinct as base dimensions. Thus Speed and Linear Productivity, both defined as specializations of Length/Time, are incompatible with each other. Unfortunately, they are both also incompatible with the dimension Length/Time. The following operations will all *fail*:

xSpeed ← xLength / xTime
xLength ← xSpeed *xTime
xSpeed + xLength / xTime
xSpeed = xLength / xTime

(Our notational convention [Section 3.3]: a variable named xA.B'C has been declared to be of dimensional type A*B/C, and a function or procedure named fA.B'C yields a result of dimensional type A*B/C.)

The quotient xLength/xTime is of type Length/Time, not Speed, which makes the first assignment above illegal.

The apparently desired results can be achieved if the user defines and applies the appropriate functions:

xSpeed ← MakeSpeed(xLength, xTime)
xLength ← MakeL'T(xSpeed) * xTime
MakeL'T(xSpeed) + xLength / xTime
xSpeed + MakeSpeed(xLength, xTime)

The MakeSpeed function would, for example, take two arguments of type Length and Time and return a result of type Speed. ***[Might want to say a little more about how that could be done.]***

8.3.2 Weakest Enforcement: Aliasing

At the other extreme, there would be no enforcement, yielding little more than aliasing. In effect, each specialization is just an alias for its parent. Two specialized dimensions are compatible if they have a common parent, which serves as their common alias. This would even allow direct assignment of one specialization to another specialization of the same parent (sibling assignment), essentially negating the value of specialization.

8.3.3 Parent/Child Assignment

One intermediate strategy would allow assignment between a specialization and its parent, in either direction. Thus, if Width and Height are specializations of Length, then the following would be permitted:

xLength $\leftarrow$ xHeight
xHeight $\leftarrow$ xLength
xLength $\leftarrow$ xWidth
xWidth $\leftarrow$ xLength

However, the basic intent of specialization, which in this case is to force a distinction between Width and Height, is still enforced by disallowing direct assignments of the form

xWidth $\leftarrow$ xHeight
xHeight $\leftarrow$ xWidth

This approach does suffer from a transitivity anomaly. The sequence

xLength $\leftarrow$ xHeight
xWidth $\leftarrow$ xLength

is valid, even though it is logically equivalent to the invalid assignment xWidth $\leftarrow$ xHeight. This can be rationalized by arguing that the above two operations represent a deliberate promotion or demotion by the user. Promoting a height to a length is a deliberate loss of information, since we no longer know which sort of length it is. Conversely, demoting a length to a height is a deliberate addition of information, like a coercion. Each of those operations done separately can be seen as making sense. However, a direct assignment of a height to a width still has the appearance of an error.

Designers of computational support systems will have to judge the merits of such a rationalization.

8.3.4 Multiplicative Promotion

Specialization often seems unnecessarily constraining in terms involving multiplication or division. This can be relaxed by allowing a specialized dimension S to be replaced by its parent in expressions of the form S*X, S/X, or X/S, if necessary for establishing compatibility.

Thus, in the dimensional expression Speed*Time, Speed could be replaced by Length/Time in order to allow cancellation, leaving Length as the result. Similarly, the expression Length/Speed could be changed to Length/(Length/Time), allowing the result to be Time.

Such substitutions (promotions) are optional, and should only be done as needed. Thus the expression (Speed*Time)/Time should yield Speed, not Length/Time as would occur if Speed was replaced prematurely. Optional promotions raise the possibility of different outcomes. The precedence of promotion relative to other operators has to be managed. Consider the dimensional expression (Speed*Time + Speed*Time)/Time. If the addition is done first, the result would be Speed, while if substitution were done first the result would be Length/Time. Further investigation is needed on this point.

8.3.5 Additive Promotion

This option almost vitiates the effect of specialization. Two specializations would be considered compatible for addition, subtraction, or comparison if they have a common parent. The result of addition or subtraction would have the type of the least common parent. Thus a height could be added to, subtracted from, or compared with a width. The result of addition or subtraction would be of type Length.

A weak form of specialization could be realized by allowing additive promotion while still excluding direct sibling assignment. That is, a height still could not be assigned directly to a width.

8.3.6 Combinations

Other variants could also be invented. The ones described above could be grouped into the following combinations, which have the sense of decreasing levels of strict enforcement:

Level	Parent/Child Assignment	Multiplicative Promotion	Additive Promotion	Sibling Assignment
5	No	No	No	No
4	Yes	No	No	No
3	Yes	Yes	No	No
2	Yes	Yes	Yes	No
1	Yes	Yes	Yes	Yes

Level 5 represents the strictest enforcement of specialization, while Level 1 is the weakest, equivalent to aliasing.

Other combinations are also possible, such as allowing multiplicative promotion without parent/child assignment, or separating comparison from addition and subtraction. Designers of computational support systems might define other configurations than these, they might choose one of these enforcement levels, or they might offer their users a choice of enforcement levels. User options might be global in scope, or they might be declared for different specializations, e.g., Speed enforced at Level 3 but Torque enforced at Level 2.

Whatever level of specialization is supported, it can be used in combination with aliasing. Users may choose to favor one sibling by defining it as an alias, with the others being specializations. Thus Speed might be defined as an alias of Length/Time, while Linear Productivity is defined as a specialization. Then Speed would conveniently be freely interchangeable with Length/Time in any context, while Linear Productivity would be protected by the specialization mechanisms. For example, Speed could not be assigned to Linear Productivity above Level 1, and they could not be added, subtracted, or compared above Level 2. Such an approach is viable to the extent that asymmetrically favoring one sibling over the others is desirable.

>>ToDo 21: Discuss the possibility and consequences of changing an alias to a specialization.

An interesting example, assuming that Area is defined as a specialization of Height*Width:

If (xHeight*xWidth)>xArea then xHeight ← xArea/xWidth.

This does not work at Level 3, which would allow the assignment, but not the comparison. It does work at Level 1 or 2.

Instead of treating violations as error conditions, another possibility would be to use these rules simply for informational feedback warnings, identifying points at which expressions are suspect and in need of review.

8.4 Multiple Parentage

Suppose we had Longness, Height and Width as three specializations of Length. We might choose to define Area as a specialization of Longness*Height and also of Longness*Width, but not of Height*Width. Such multiple parentage creates a number of problems.

When any sort of promotion is supported [Section 8.3], multiple parents are available to be substituted. Thus, for example, Area/Longness could turn out to be either Height or Width. Some contexts provide a criterion for selecting one of these, e.g., if Area/Longness was being added to either Height or Width. However, there are also ambiguous cases. Suppose there were another dimension B which was a specialization of Time*Height and also Time*Width. Then the expression

xArea/xLongness + xB/xTime

would be ambiguous, yielding either Height or Width.

There are several possible approaches to such ambiguity:

- The sledgehammer: disallow multiple inheritance because of this problem.
- Legislate that such ambiguous expressions are invalid.
- Introduce some notion of "type sets", trying to see whether the ambiguities are resolved by the end of the expression evaluation. The type of a sum might be the intersection of the types of its terms. Thus, the result of that example would have type {Width,Height}; if it was then added to xWidth, the ambiguity would be resolved.

Further investigation is needed into the desirability and feasibility of multiple parentage.

8.5 Multi-Level Specializations

[Illustrate. Can we find some interesting ones which don't descend from Unary? Area under Width and Height? Volume specialized from Area? Simpler examples based on single parentage?]

>>ToDo 22: Examples of multi-parent specialization?

8.6 The Unary Dimension and its Specializations

8.6.1 Purpose

All multiplicative combinations of base dimensions, algebraically reduced, constitute distinct dimensions.

The *unary dimension* collects the set of dimensions that appear to be dimensionless, e.g., angle (equivalent to Length/Length), certain concentrations such as Volume/Volume, clock rate error (Time/Time), and so on. As the examples illustrate, these all take the form of ratios reducible to 1. While counts appear to be another category of "dimensionless" dimensions, we don't see a need to include them among the unaries for now.

The unary dimension is the product of no dimensions, i.e., the product of all base and specialized dimensions to the zero power. Hence its signature is 0. It cannot itself be a base dimension; otherwise its signature would have a 1 in it. Unary is the result type of an expression when all the dimensions cancel out.

8.6.2 Implicit Specializations

The "first tier" of specializations of Unary is an implicitly-defined ordered infinite set corresponding to unit ratios of base dimensions. For example, the first of these might correspond to Time/Time. While this dimension "corresponds" to Time/Time, it is not the same thing, so we use $\overline{\text{Time/Time}}$ to denote this specialization of Unary. One important distinction is that the specializations are not algebraically reduced. Thus, while Time/Time and $Time^2/Time^2$ would be the same thing, $\overline{\text{Time/Time}}$ and $\overline{\text{Time}^2/\text{Time}^2}$ are not. In fact, Time/Time is the same as Length/Length, but $\overline{\text{Time/Time}}$ and $\overline{\text{Length/Length}}$ are not — which is precisely why we introduced unaries in the first place.

8.6.3 Explicit Specialization

>>ToDo 23: Notes. To be completed...

Angle would be a specialization of $\overline{\text{Length/Length}}$. And that might expand out into a whole complex of various flavors of angles.

$\overline{\text{Volume/Volume}}$ leads to some interesting things:

- It is a specialization of Unary.
- It seems to involve a specialization (Volume).
- It could have two further sub-specializations: one for solute/solution, one for solute/solvent.
- Those two sub-specializations are members of the quasi-dimension Concentration.

What a graph!!!

8.6.4 Signatures

[Say more about the concepts before diving into notation. Some of this refers forward to Section 8.7.]

For the purpose of defining signatures, we observe that these implicitly defined unaries constitute an orderable infinite set. We assume some arbitrary ordering. We have three kinds of dimensions which require positions in a signature:

- Base dimensions (finite number = b).
- Explicit specializations, including explicit specializations under Unary (finite number = s).
- Implicit specializations of Unary (infinite).

The infinite set needs to come last in the signature notation. Hence signatures are constructed as follows:

- The first b positions represent base dimensions.
- The next s positions represent explicit specializations.
- Remaining positions represent the implicit specializations of Unary.

Thus, for example, $\overline{\text{Time/Time}}$ (the first implicit specialization of Unary) has signature $0^{b+s}1$.

This is clearly only useful as an expository notation. Signatures of all dimensions are subject to change as new base or specialized dimensions are introduced.

8.6.5 Units

The importance of these implicitly defined unaries is that they impart units to their specializations. Thus, while Clock Rate Error is apparently "dimensionless", it has units of the form Time-unit/Time-unit (e.g., seconds/year) but not of the form Length-unit/Length-unit. Similarly, Strain has units of the form Length-unit/Length-unit (e.g., inches/mile). These definitions are obtained by specifying Clock Rate Error as a specialization of $\overline{\text{Time/Time}}$ and Strain as a specialization of $\overline{\text{Length/Length}}$.

As with other specializations, additional units can be introduced for the specialization, such as degrees for angles.

There is also an anomalous "unitless" form when the numerator and denominator are both expressed in the same units — in which case it is simply a ratio, independent of any particular units. Thus, for example, a Clock Rate Error of 0.01 means 1 second/100 seconds or 1 hour/100 hours or ...

>>ToDo 24: PQE's for dimensionless and composite dimensions?

[Does this mean we have to allow "unitless" PQE's? How do we know the dimension involved? Actually, the whole subject of PQE's for composite dimensions remains to be clarified.]

>>ToDo 25: Is percent a unit?

[We also raised the question of whether "percent" is a unit, or simply a different notation for rational numbers.]

8.7 Specialized Canonical Form

Specialized canonical form extends techniques of traditional dimensional analysis [Section 4] to deal with specialized dimensions.

For purposes of canonical form, specializations are treated much like base dimensions. The canonical form of Speed is simply Speed, not Length/Time. Thus each specialization gets a distinct new signature, not a combination of previously defined signatures (Figure 1). Our shorthand for signatures is augmented with the construct 0^b representing b zeroes, where b is the number of base dimensions. At the moment b=3 since we have three base dimensions defined (in Figure 1). Longness, the first specialized dimension, therefore has a signature of 0^b1, i.e., 0001.

For the sake of illustration, we have also introduced Distance (in the sense of distance travelled) as a distinct specialization of Length. It will be used in later examples.

>>ToDo 26: Should we mention alternate encoding of signature?

[Mention the option of encoding the parent in the prefix, under single inheritance. Expedites checking, since specializations can never be compatible unless they have a common parent.]

8.8 Other Experiments With Specialization

8.8.1 Another Approach to Work and Torque

Suppose we took advantage of Distance as a specialization of Length, in the sense of distance travelled.

To begin with, Speed and Acceleration should then really be defined as specializations of Distance/Time and Distance/$Time^2$. (Here I do really mean "Distance", the specialization.)

The first mind-blower: where do multiplicative combinations of base *and/or specialized* dimensions fit in the type graph (Figure 1)? That may be trivial, once I get used to it, but I'm not so sure just yet.

Force should then be defined as a specialization of Mass*Acceleration, not just ML/T^2.

Since Work has the sense of a force being applied to a body in order to move the body a certain distance, it would naturally be defined as (a specialization of?) Force*Distance.

Torque has the sense of a force being applied statically around a point of rotation, at a radius of a given Longness(!). Therefore Torque could sneakily be defined as (a specialization of?) Force*Longness.

The good news is that we've now made Work and Torque incompatible by defining them in terms of different specializations of Length.

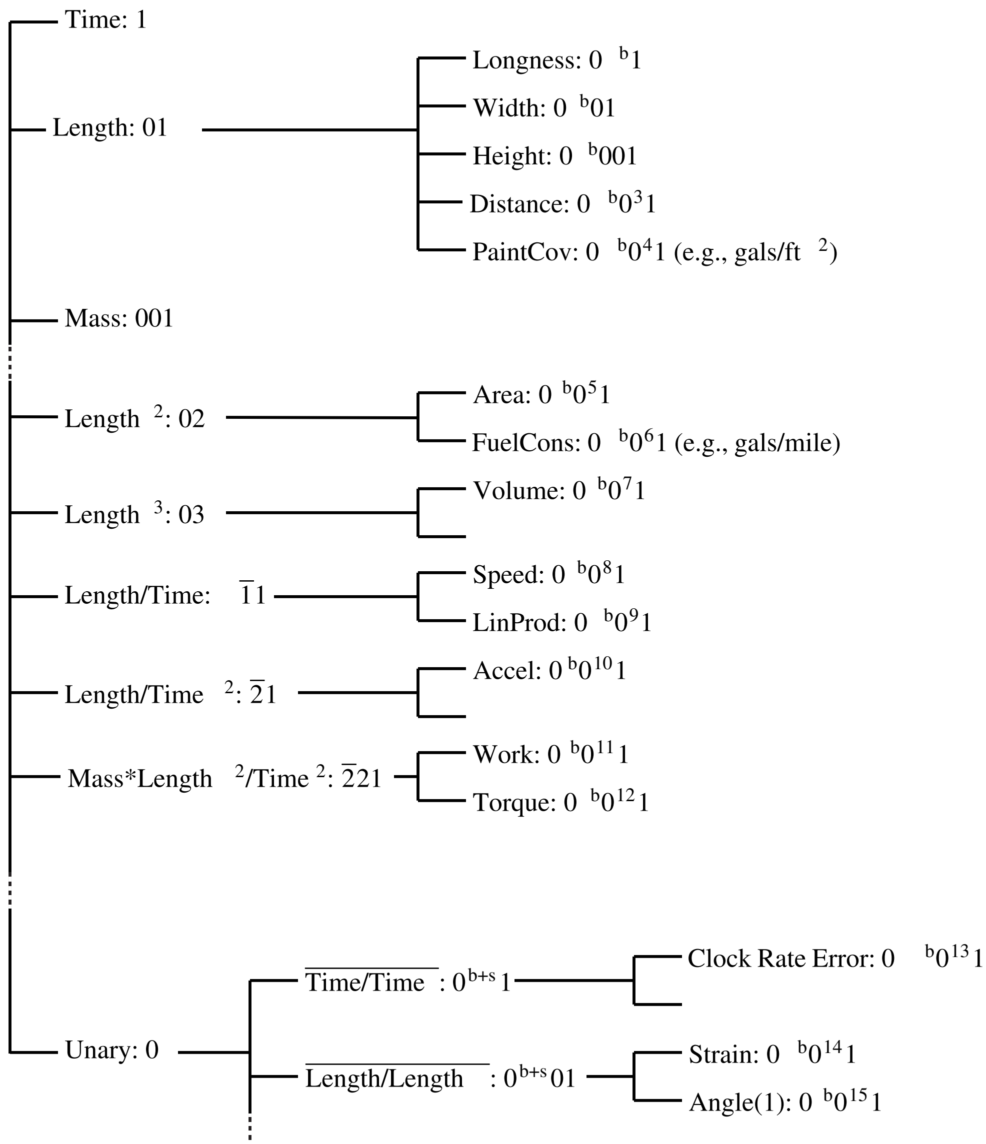


Figure 1.

The bad news is that I don’t quite see how to convert Torque to Work by multiplying by an angle. It is the intuitive transform: maintaining a certain torque while moving through a certain angle of rotation corresponds to a certain amount of work. It seems to need a definition of Rotational Angle as Distance/ Longness. If we could tolerate that, then we would get a nice formulation:

Torque * RotAngle = Work

Force*Longness * Distance/Longness = Force*Distance.

Is there any hope of something useful here?

Would it be too crazy to even introduce Radius as yet another specialization of Length, and use that instead of Longness above? It does seem to capture the semantics involved in both Torque and Angle.

The down side, as always, comes from having to explicitly force Radius to be a Length when that's what you really want. Oh well, the trade-offs.

9 Generalization (Quasi-Dimensions)

>>ToDo 27: Possible application to currency conversion?

[New observation (5/17 wk): having to fix the conversion parameters in the current context seems to bear some resemblance to fixing currency conversion rates.]

[Potentially another quasi-dimension in the travel example! "Travel segment" can be expressed as either Time or Distance. Made compatible if a Speed parameter is fixed. AAA maps do that. Also astronomy. Dwell on this analogy a bit longer. One set of factors converts between units of a given dimension. Another set of factors converts between member dimensions of a given quasi-dimension.]

In Section 8 we dealt with the case where many dimensions have one canonical representation. Here we address the converse: one dimension having many canonical representations.

Consider the fact that concentrations can be expressed in Volume/Volume, Weight/Weight, and Weight/ Volume. Concentration has some of the characteristics of a single dimension:

- It corresponds to the single semantic concept of relative amounts of two things.
- Concentrations of the same material may be expressed in different terms in different situations, and it is natural to convert among these forms in much the same way as units conversion is done.

Yet concentration is not a single dimension in the traditional sense, since it corresponds to several distinct canonical forms. We might call concentration a "quasi-dimension." It is a single abstraction, or generalization, of several distinct dimensions.

Many of the quasi-dimensions we have identified so far are based on the underlying quasi-dimension of "Amount." Amounts of things can be expressed in terms of Length (e.g., rope), Area (e.g., carpet), Volume, or Mass. Amounts of medication are often expressed in the number of pills or drops. Sometimes (e.g., when counting populations) we leave off the thing being counted and just mention the number. With "Amount" established, concentration can then be expressed as Amount/Amount. Other such quasi-dimensions include Price (Money/Amount) and Dosage (Amount/ Time).

Other quasi-dimensions are not based on "amount." For example, pressure can be measured in pounds/ square inch (Force/Area), or as millimeters of mercury (Length). Similarly, Length could be measured in feet (a length unit), or in years (a time unit), assuming the speed of light. Both of these quasi-dimensions (Pressure and Length) thus have more than one canonical representation. In the audio and radio domains, time-domain signals and frequency-domain signal spectra are equivalent representations of the same thing. They have representations of Amount/Second and Amount/Hz (or Amount/octave) and are inter-convertible.

Rewrite the following for continuity and logical flow.

There are a few side issues, related to concentrations and similar quantities, which do not bear on the subject of quasi-dimensions. Concentration is sometimes specified as solute/solvent and sometimes as solute/solution. ***[See discussion in Section 8.6.3.]*** This is not the same issue as which dimension to use to represent a particular concentration. It is more like the choice of unit (relative vs. absolute pressure) or a detail in the choice of quantity measured (like inside diameter vs. outside diameter). We are not sure which of these is the proper model for the distinction, but each is well-understood. We do not discuss this distinction further, and instead focus on the dimensional issues. The density of solvent or solute (which is needed for conversion) depends upon whether it is solid, liquid or gas. We do not worry about that here, but assume that any mechanism for choosing the density of a material can choose the density of the proper state of the material. When solutions are combined physically, the resulting concentration may not be a simple function of the initial concentrations. We do not worry about the physics or chemistry of combining solutions. We confine ourselves here to the relationship between concentration and the canonical forms used to represent it.

Support for quasi-dimensions involves defining them, establishing their compatibility rules, and automatically converting among compatible canonical forms.

9.1 General Approach

>>ToDo 28: To be completed. These are just notes.

The general approach has two main parts: defining convertibility graphs, and establishing their scope.

A convertibility graph for members of a quasi-dimension would have much the same characteristics as the conversion graph for units of a given dimension. There has to be:

- Definition of which dimensions are members of the quasi-dimension.
- Specification of conversion factors (procedures?) between pairs of member dimensions. Some of these are simple constant properties, such as the density of a substance. They might be more complex.
- Verification of conversion paths between any two members.
- Tactical choices among various possible conversion strategies.
- Concerns about accuracy and precision in conversion.

Scope is a much harder new issue. Obviously the same graph doesn't apply to all cases. In fact, the same quasi-dimension would often have multiple convertibility graphs, e.g., for different substances. It's probably necessary to name these graphs.

Graphs might be dependent on other things than the substance involved. Some graphs might reflect the assumption of sea-level atmospheric pressure, or of STP (Standard Temperature and Pressure), or Earth's gravity. Sometimes the "substance" might not be more specifically identified than that it is a gas, since certain laws of physics are uniform for all gases.

The scoping problem has the usual generic form and options. The essential point is to establish which convertibility graph, if any, is in effect at a particular moment. There is the usual range of solutions, including:

- Globally fixed for all time. Not practical.
- Make it a property of the procedure/module/program.
- Explicit declarations in the user's program to Begin and End a scope.
- Specify the appropriate conversion graph right with the expression.
- Figure out some way to know the substance involved from the data in the environment.
- Others?

9.2 Compatibility

Part of the meaning of a quasi-dimension is that it makes sense to convert quantities among its canonical forms, where such conversion would not be legal otherwise. The methods for performing conversions are application-dependent (see below), and must be provided by users or administrators. A system has several choices for deciding on the legality of conversions.

- It can say that all conversions are legal (and thus require a method for every possible one when a quasi-dimension is defined).
- It can say that the legal conversions are exactly those which it has been told how to do.
- It can specify the legal conversions algorithmically (e.g., Volume and Mass are interconvertible only for temperatures at which water is a liquid).

These compatibility rules combine so as to add compatibility: when a dimension joins a quasi-dimension, it thereby becomes potentially compatible with more dimensions. It cannot thereby become compatible with fewer of them.

***>>ToDo 29:* Compatibility question.**

When defining a quasi-dimension, do we get to override any preexisting compatibility rules? For example, can we make any of the children incompatible with things they used to be compatible with?

***>>ToDo 30:* What about compatibility of multiplicative combinations?**

Is mass/vol*charge compatible with vol/vol*charge? By analogy with specialized dimensions, they would lose their "quasi-ness."

9.3 Converting Among Canonical Forms

Converting between different canonical forms often requires knowing some property of the substance in question, such as the density of the solvent or solute. Such conversions are *substance-dependent*. In some situations, a system can't be expected to know the required information. In these cases it is reasonable to require a user to describer explicit conversions. For example, to convert from a volume v to a mass m, the user could write

$$\text{mass} \leftarrow \text{vol} \times \text{k grams/cc},$$

explicitly specifying the density of the substance.

In some situations automatic conversions are possible, but require an application-dependent conversion. It is often appropriate to assume that the substance has properties similar to water. Alternatively, in certain industrial settings there is only one substance involved in all of the data. Nurses are routinely expected to translate dosages into drips/minute for intravenous solutions. For these situations, the user or administrator must provide either the appropriate conversion factor (assuming it is multiplicative), or the entire conversion routine. Conversions can be either pairwise or a "star" configuration [Section 29.3.1]. Defaults may be provided for conversion factors [Section 24.3].

Other conversions are easy to do automatically, such as converting from miles/gallon to gallons/mile.

10 Summary of Inter-Dimension Relations

***>>ToDo 31:* To be completed. Are there others?**

III MEASUREMENT PARADIGMS

What sorts of measurements can be supported?

>>ToDo 32: To be completed.

Extensibility. Axioms to test what sorts of measurements can be introduced, and how defined.

Axioms can establish that a measurement domain is not supportable, or that it should be supported in a particular style [Section 11.2].

We first discuss the units-based paradigm and its variants, and then other measurement paradigms which would not be supported by an implementation of the units-based paradigm.

11 The Units-Based Paradigm

We will describe in painful detail the characteristics of one well-behaved measurement domain, namely Length. ***[I picked that rather than weight to avoid discussions of its relation to mass. I could go back to using weight.]*** The significance of each characteristic will be illustrated by comparison with some domains that behave differently. Afterward we can try to systematically characterize other sorts of domains along similar lines.

It's important to always keep in mind that we are discussing a concept other than numbers. We must be especially careful not to assume any property for this domain just because it happens to hold for numbers. Length is some mysterious notion which exists and has certain behaviors long before we get around to measuring it and denoting it with numbers. We explore the properties of lengths, not numbers, precisely for the purpose of establishing when and how the behavior of numbers provides a correct model of the behavior of lengths.

The goal is to characterize a "units-based" measurement domain in the following sense: given two lengths L_1 and L_2, there exists a unique real number k such that $L_1=kL_2$. The length L_2 serves as the unit for the measurement of L_1. There's a surprisingly large number of underlying assumptions that have to be verified about the measurement domain before this paradigm will hold.

[Proofreading nuisance: I seem to wander between using L_i and x, y, z for lengths.]

11.1 Fundamental Properties

11.1.1 We Know What It Is

To begin with, we have reasonably precise consensus on exactly what we mean by length, even if we have struggle to articulate a precise definition. Certain modern physicists might not agree here, but the context of Euclidean geometry is sufficient for our present purposes.

In contrast, we don't have any such clarity on what exactly is meant by beauty, intelligence, sports performance, etc.

A slight embellishment of this point (is it a distinct point?) is that we can recognize the individuals as well as the collective concept. That is, there seems to be no real problem in agreeing whether a particular thing is a length or not. (This will be slightly challenged when we come to negative lengths.)

11.1.2 We Can Tell Them Apart

The concept of whether two lengths are the same or not is also sufficiently clear. This is an idealized principle, neglecting concerns of precision and accuracy of measurement. Thus we can postulate a simple, total *equality* comparison operator under which any two lengths are either equal or not. We arbitrarily postulate our intuitive belief that this operation is reflexive (x=x), symmetric (if x=y then y=x), and transitive (if x=y and y=z then x=z). Hence we may safely use the familiar = to denote this comparison.

Contrasts arise in the same domains mentioned above. There is also such a problem with angles: is the angle denoted by 360° the same as the angle denoted by 0°?

11.1.3 There Is Total Order

The notion of relative size is also sufficiently clear. We can safely postulate a total ordering under which, for any two lengths, either they are equal or exactly one is smaller than the other. It seems safe to use the familiar <, > symbols to denote ordering comparisons. (We postulate that the axioms of order are satisfied, e.g., transitivity, anti-symmetry, anti-reflexivity.)

Same contrasts, including angles. 40°=400°, being simultaneously less than and greater than 100°.

11.1.4 There Is Combinational Closure

There is a well-understood concept of combining two lengths to yield another length, intuitively corresponding to the notion of laying them end to end. Combining any two lengths in this way always yields another length (the closure property).

While this has the sense of addition, we will denote this combinational operation by # rather than + to emphasize that we are *not* yet talking about numbers, and hence not about arithmetic. The relevant behaviors still need to be established.

Each domain potentially has its own combining operation which serves as the basis of the measurement paradigm. For every x and y in a domain D, x#y must exist, and must be a member of that domain.

There is no sense of order in the combination process. Our linear notation requires that we write one participant before the other, but x#y means the same thing as y#x. We thus have

- *Commutativity*: x#y=y#x.

Order-independence is more than notational. If we first combine x and y, and then combine the result with z, we get the same end result as combining x with the result of combining y with z. We thus have

- *Associativity*: (x#y)#z=x#(y#z).

This is a non-trivial requirement which might not be satisfied, for example, when mixing chemicals. If we are to model # with the arithmetic of addition, x#y must be commutative and associative.

x#y must be repeatable: it must yield the same result for all instances x with equal magnitude, and for all such instances y. This property is not always trivially satisfied: pouring a cup of salt into a cup of water does not yield the same volume as pouring a cup of water into a cup of oil—combining volumes is more than simply pouring them together. Similarly, the combining operator must depend only on x and y. The result of combining two masses must not depend on the temperatures of the bodies whose masses are being combined, and the result of combining temperatures must not depend on the masses of the bodies. Combining two lengths must not depend on the angle between them, and combining two angles must not depend on the lengths of the arms subtending the angles.

Summarizing, x#y must satisfy:

- Totality: x#y exists for any x, y in D.
- Closure: x#y is itself an instance of D.
- Repeatability: $x_1=x_2$, $y_1=y_2 \Rightarrow x_1\#y_1 = x_2\#y_2$
- Commutativity: x#y = y#x
- Associativity: (x#y)#z = x#(y#z)

>>ToDo 33: Counterexamples?

[Are there contrasting domains in which such an operation does not exist?]

11.1.5 Monotonic Combination

For lengths, $x\#y \geq x$. This doesn't always hold, even in totally ordered domains, such as circular angles [Section 7.1.2].

11.1.6 There Is A Natural Nil

There exists a length $\varnothing$ such that $x\#\varnothing=x$ for any length x.

This nil length is unique. If $x\#\varnothing_1=x$ and $y\#\varnothing_2=y$, then $\varnothing_1=\varnothing_2$.

>>ToDo 34: Is uniqueness of nil an independent axiom?

11.1.7 There Are Differences

Combinational closure assures us that if x and y are lengths, then x#y is a length. It doesn't guarantee that there exists a length z corresponding to the difference between x and y. This has to be postulated independently, and it comes in two flavors, absolute difference and subtraction.

Absolute difference: for any two lengths x and y, there exists a unique $z \geq \varnothing$ such that either x#z=y or y#z=x. This will be denoted $x \Delta y=z$.

It should be provable that $x \Delta y=y \Delta x$.

11.1.8 Subtraction: There May Or May Not Be Negatives

If x>y, is there a length z for which x#z=y? In particular, if $x>\varnothing$, is there a length z for which $x\#z=\varnothing$? Yes and no.

Sometimes we want (3 feet − 4 feet) to be an error condition, and sometimes it is a legitimate negative length (e.g., tides, or net yardage in football). An appropriate solution is to postulate two domains (dimensions, types) corresponding to signed and unsigned lengths [Section 5].

We will use y~x to denote subtraction among lengths. For signed lengths, y~x always exists, and is the length z such that x#z=y. For unsigned lengths, y~x is an error if y<x, otherwise it yields the same result as for signed lengths. If $y \geq x$, then $y \sim x=y \Delta x$.

>>ToDo 35: Does any of that need to be proved?

11.1.9 Multiplication and L-Rational Lengths

Associativity and commutativity of #, together with combinational closure, insure that any collection of lengths can be combined in any order to yield the same result. In particular, the result of x#x#…#x, involving k instances of x, is a unique length, which we can denote k*x or kx or x*k. We can arbitrarily define $0x=\varnothing$, and observe that $k\varnothing=\varnothing$ for any k.

We have thus established that kx is a meaningful expression for any length x and any non-negative integer k. Note that this is the first formal occurrence of numbers in this development. Previous postulates have concerned lengths, not numbers.

Uniqueness of k is important and has to be independently postulated, i.e., for $x \neq \varnothing$, if $k_1x=k_2x$ then $k_1=k_2$. This axiom does not hold, for example, for circular angles [Section 7.1.2]. If x is a 1° angle, then x=361x=721x …

Exercise: verify that $k_1x\#k_2x=(k_1+k_2)x$, and also $k_1(k_2x)=(k_1k_2)x$. Does that follow from commutativity and/or associativity of #? *[Relates to Section 34.12].*

We can introduce rational numbers as follows: if $k_1 x_1 = k_2 x_2$, $k_1 \neq 0$, we then say that $x_1 = (k_2/k_1)x_2$. The rational number k_2/k_1 is unique for given lengths x_1 and x_2.

For a given length L, we will say that the *L-rational* lengths are those which are equal to rL for some rational number r. The set of such rational lengths clearly depends on the choice of L. It is intuitively evident that a length which is rational relative to a one-inch length is irrational with respect to a length which measures π inches. Similarly, for angles, those which are rational relative to an angle of one degree are irrational with respect to an angle of one radian, and vice versa.

Observe: if L_1 is L_2-rational, then L_2 is L_1-rational.

11.1.10 Continuity and L-Reachable Lengths

If all lengths were rational with respect to each other, we would have a sufficient foundation for units-based measurement. Any length L could be chosen as a unit, and any other length would be equal to rL for some rational number r.

However, lengths are not all rational with respect to each other. We could extend the set of lengths measurable using L if we allowed r to be a real number in the expression rL. However, we haven't yet defined what rL means for irrational r.

Mimicking the development of irrational numbers, an irrational length is one which is not rational but can be approximated to any arbitrary degree of closeness by rational lengths. Remember that we assumed that the notions of difference and relative size among lengths are well-defined for all lengths, rational or not. All we need now is an *axiom of continuity*:

> For any three lengths x_1, x_2, x_3, where $x_2 \neq \varnothing$, $x_3 > \varnothing$, there exists a rational number r such that $x_1 \Delta r x_2 < x_3$.

If we are trying to measure x_1 using x_2 as a unit, this axiom says that rx_2 is a good approximation of x_1, differing from x_1 by at most x_3, which we can initially choose to be as small a non-negative length as we want. This has to be postulated as an independent axiom for a domain such as Length.

>>ToDo 36: Further proof needed?

[Do we need one more step to justify the use of irrational numbers to measure irrational lengths?]

It may be sufficient to reason only about the L-rational lengths. Problems with equality? Uniqueness of measure? (Domains which are representable by bit strings are totally L-rational. May be able to reason interestingly about the domain of representable colors.)

11.1.11 Coverage

In order for something to be a unit, every instance of the domain must be "reachable", i.e., expressible in terms of the unit. For a unit u, any instance x must equal ru for some number r.

For a given length L, we have extended the set of "reachable" lengths to those which are expressible as rL for some real number r. An axiomatic property of lengths is that all lengths are reachable as rL for any length L.

This is not true in all domains. It fails for vector dimensions such as velocity. There is no velocity V such that any velocity can be expressed as rV.

This probably also explains why colors can't be measured with a simple unit. I.e., there does not seem to be a single color C and a single combining operation # under which any color is equal to rC for some rational number r.

11.1.12 Degenerate Units

In general, a domain is units-measurable if the domain has a 1:1 correspondence onto the rational (or real) numbers. This implies some mapping M from the domain onto the numbers.

Our previous analysis has based such a mapping on some intuitively natural operation on the underlying quantities, such as laying lengths end-to-end or volumes side-by-side. For colors, we'd like to imagine some natural operation such as mixing of some sort.

However, such "natural" operations are not necessary. If there exists some sort of mapping, perhaps arbitrary, from a domain to the numbers, which satisfies the necessary axioms, then the domain is unit measurable. We reverse the previous development by saying that x#y=z if M(x)+M(y)=M(z), and so on. Any element x can serve as a unit, such that for any y in the domain, y=[M(x)/M(y)]x.

There might be subsets of certain domains, such as computer-representable colors, which can be measured in this sense.

>>ToDo 37: To be investigated.

11.2 Variants Within the Units-Based Paradigm

>>ToDo 38: To be completed.

This is where we can talk about variations which might be supported within the units-based paradigm. Presumably correspond to some variants of the axioms developed above.

E.g., ordinary addition vs. modulo addition, point vs. interval, others.

Exponentials, such as dB, pH, and Richter scale, probably also come here. Observe: if there is no addition, is there subtraction? How do differences work?

> ***[Again, need to wrap up with some sort of conclusions, as well as a capsule summary of the axioms and their uses.]***

12 Vector Dimensions

>>ToDo 39: More to be said?

Some dimensions, such as velocity, field strength, locations in n-dimensional space, etc., are well-modelled by the mathematical notion of a vector. For vector operations to yield intuitive results on an instance of a dimension, it should have these properties:

- It is a single physical quantity. Collections of related quantities are discussed in Section 13.
- It requires more than one number to represent it. The numbers must be *required*: constructs like, "5 feet, 5 inches," used to mean 65 inches, do not qualify, since they can be replaced by a single number (these are discussed in Section 19.2). Similarly, a speed of "5 miles in 2 hours" can be replace by "2.5 miles/hour." 1-element vectors are the same as scalars, and we don't discuss them here.
- Vector quantities are equal if and only if they are equal under element-by-element comparison. This again disqualifies speed described a "5 miles in 2 hours," since it is equal to "10 miles in 4 hours."
- A vector with a different number of elements belongs to a different dimension. Thus, a location in two-dimensional space is a different dimension from a location in three-dimensional space.

Section 19.4 and Section 26.3 describe ways to represent and manipulate vector physical quantities. We confine the discussion to 1-dimensional vectors whose elements are themselves scalar physical quantities. These restrictions are not required in principal, but they simplify the discussion, and our intuition hasn't shown us any reason to remove them.

>>ToDo 40: Stephanie's questions:

Question: Should vector dimensions be in the type hierarchy?

I know we originally decided that they should be (rather than having just scalar dimensions, and then a Vector type), but I thought we might reconsider this, especially as we are thinking of keeping Aggregate out of the type hierarchy. There seems to be some inconsistency here, and I'm not sure how to resolve it.

I still do think that vector dimensions belong in the type hierarchy, with similar arguments as I used for the null dimension. If a vector dimension isn't in the type hierarchy, then all specializations of what we are calling vector dimensions must be done component-wise. This means that <time,time> will always be compatible with <time,time> This is probably not flexible enough. (We could always product the effect of vector specializations by creating specialization of the components just for their vector use, but this seems sort of kludgy to me. The ideal solution seems to be to allow the entire vector to be specialized, this implying that vector dimensions are in the type hierarchy.)

Question: Is the assumption of considering derived dimensions to be multiplicative combinations of only scalar dimensions too restrictive? What about considering vector dimensions to contain only scalar dimensions as components?

Proposal: Vector dimensions are made up of an order-dependent list of dimensions. These dimensions can be regular, specialized, or vector dimensions. (Note: they can't be quasi-dimensions, as we are not currently considering quasi-dimensions to be "true" dimensions. This seems strange, as <time,concentration> seems like a perfectly good vector dimension to me. This needs discussion.)

A derived dimension is a "mathematically legal" multiplicative combination of dimensions. For example

time<distance,mass>

is legal, but

$$\frac{\text{time}}{\text{distance, mass}}$$

is not, as the inverse of a vector doesn't make sense. Similarly,

<dim1, dim2><dim3, dim4>

is not legal, as dim1*dim3 + dim2*dim4 is not a legal dimension. (I think that the rule is that only scalar multiples of vectors are OK, and the scalar can be made up of an arbitrarily fancy dimensional expression.)

Question: Can the functions Dimension, Unit and PhysicalQuantity be called on both SPQEs and VPQEs, or can these functions only be called on SPQEs and components of VPQEs?

Related questions: is a vector dimension uniquely determined by its associated tuple of dimensions? For example, (width, height) and (x, y) are both (length, length), but it may not be the case that they should be treated as compatible (in the same vector dimension). Should the representation of a vector dimension be a tuple of

scalar dimensions or a single vector dimension? If a VPQE is represented as a tuple of SPQEs, how do we know that its dimension is velocity (rather than <meters/sec, heading>)?

I had been thinking that the dimension of a vector dimension would be the vector of the dimensions of its components. However, this is a direct contradiction to being able to specialize a vector dimension. I don't know how to reconcile this. I think this is the most important open issue for vector dimensions.

13 Aggregates

>>ToDo 41: More to be said?

[Stephanie:] Question: Are we going to pursue what is needed to support aggregate measurements? Should aggregates be in the type hierarchy?

The answer to this may depend on the answer for vectors, as they are a special case of an aggregate type. How does OpenODB handle this? (I think vectors should be in the type hierarchy.)

An aggregate of measurements is a set of measurements organized in some way. For example, a waveform consists of a number of readings taken over time. Similarly, a set of measurements might be associated with a grid, where there is one measurement at each Cartesian coordinate (given some granularity). The boundary between aggregates and non-aggregates is not always clear. For example, a waveform can also be represented as a number of individual measurements, each of which carries time information as part of its context.

The notion of aggregate is loosely-defined: as suggested above it could mean variously a set, list, grid, or uniformly-spaced grid, and can have any number of dimensions. We therefore do not try to say specifically what is involved in supporting aggregates, but we make a few characterizations.

- It is necessary to define them, of course. The elements being aggregated must all be of the same type, but the aggregate may also contain information of other types (such as the time duration between voltage samples).
- It is desirable to be able to add and remove members.
- The arithmetic operations and compatibility rules need to be specified. They may simply be the usual rules applied element-by-element, but they may be much more complex, as in addition of waveforms whose sample times do not coincide.
- The "value" of an aggregate may be defined. It will often be the arithmetic mean of the members, but it could be the mode, the maximum, or any other property. An aggregate need not have a "value."
- Aggregates may have properties which the individual members do not, such as:
  - Number of elements
  - Spacing between elements
  - Statistical properties of the distribution of the elements: moments, bounds, etc.

  These can be computed automatically, but the notion of which ones are relevant must be provided in the definition of the aggregate.

One interesting kind of aggregate omits the elements, and mentions only the statistical properties (we call this a *virtual aggregate*). When we speak of the temperature of a glass of water, we mean the temperature of many water molecules, all assumed to be at approximately the same temperature.

When we speak of the last hour's wind speed, we mean not only the speed of many air molecules moving past our measuring device, but also the aggregate of many speeds during that hour. Note that we sometimes give an average to describe wind speed, and sometimes a mode and a maximum, as in "winds of 20 m.p.h., gusting to 50." If we speak of the temperature of the Pacific Ocean, we now mean the aggregate over many points in the ocean's volume (where each point is itself an aggregate similar to the glass of water). This sort of aggregate—with the elements omitted—can thus be taken over number, time, and space.

Note that the examples above have exact values. There is an average of the temperatures of all the molecules in a glass of water at an instant; there is an exact average, over all air molecules, of all their speeds in the past hour; and there is an average temperature of all the water molecules in the Pacific Ocean at an instant. People rarely mean such things in practice, and it would be infeasible to compute them. In the examples cited, if we actually reported such a value, it would be distinguishable by its extraordinary precision. Nevertheless, these examples reveal an ambiguity in using statistical properties to describe an aggregate. This ambiguity is already present, of course, and each discipline's common practice has found ways to work around it. A system supporting physical quantities need not try to resolve these ambiguities, but it should be prepared to deal with virtual aggregates. There are three ways to do this:

- One can develop a separate mechanism to represent them.
- One can use the existing mechanism for ordinary aggregates, but omit the elements.
- One can represent the value of a virtual aggregate with a scalar, and carry the statistical properties as context.

14 Enumerations

Question: Are enumerations units or dimensions?

Solution: Enumerations (such as grades) are the range of units. They are generally coarse-grained over a continuous domain.

***>>ToDo 42:* Does enumeration correspond to a different measurement paradigm?**

[Speculation: that constitutes another sort of measurement paradigm. If not, then maybe this stuff should get moved over to Units. Maybe it should be mentioned there anyhow.]

15 Non-Units-Based Measurement

***>>ToDo 43:* To be completed.**

This is where we might talk about other paradigms such as testing, polling, judging, and formulas (statistics?) (e.g., batting averages, quarterback ratings).

Things like intelligence, beauty contests, sports performance, computer performance (efficiency), complexity, usability, etc. all come in here.

The general thesis: units-based paradigms don't apply, hence some other paradigm is required as a basis for computational support.

15.1 Counts

Notes...

In one sense, units-based measurement is nothing but counting: how many units is this magnitude equal to? Also, is it still counting when fractions are involved? We can probably contrive some examples. Fractional frequencies, average populations or committee sizes. ("The average size of the committee over the past year is 4.8" .)

Counting is the limiting case of units-based measurement, when there is just one natural and self-evident. The unit happens to be something that we perceive as a natural whole entity, unique in its domain. Units-based measurement would look more like counting if we didn't have to mention the unit, if it was a self-evident default. (Examples: populations, committee sizes. If someone said the committee size is nine, you wouldn't ask "nine what?". Distance might be the same if it was always in meters. Highway signs often look like counts because the units are implicit.)

>>ToDo 44: Open questions about counts and frequency:

- How would we characterize frequency? As 1/time? As unary/time? As count/time? Is there a difference?

  For now, we will characterize frequency as 1/time.

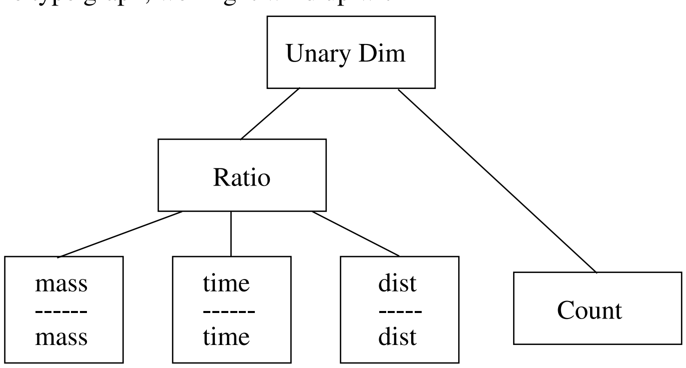
- Where does count go in the hierarchy? Consider count/time.

  We are not considering count as a specialization of the unary dimension for now becuase we can specialize from things like 1/time. (We don't need count-spec/time to get the effect we want.) Part of the problem of putting count in the hierarchy as a specialization of the unary dimension is that we get into the discussion of "count of what?". For example, is a count of population compatible with a count of cycles? It is counter-intuitive [pun?] to allow things like birthrate and electrical frequency to be combined, however, they both have a dimension of 1/time. This could be handled by specializing 1/time. We currently think that this is a better solution than adding count as a specialization of the unary dimension, and ending up with things like people-count/time and cycle-count/time. (Messy issue - making specializations based on "count of what".) We may need to revisit this when we look at specializations and the type hierarchy.

- How do we position count in the graph so that it can belong to the "dosage rate" quasi-dimension?

  We might need to convert cc's or milligrams into capsules or pills, and cc/min into drops/min.

If we did put Count into the type graph, we might wind up with



16 Generalized Type Graph

>>ToDo 45: To be completed.

Now that we have broadened what we mean by "dimension," it is time to revisit our type hierarchy. How has it changed?

- Quasi-dimensions are ways of organizing existing dimensions, but are not themselves dimensions. They don't add anything to the type graph, nor remove anything from it.

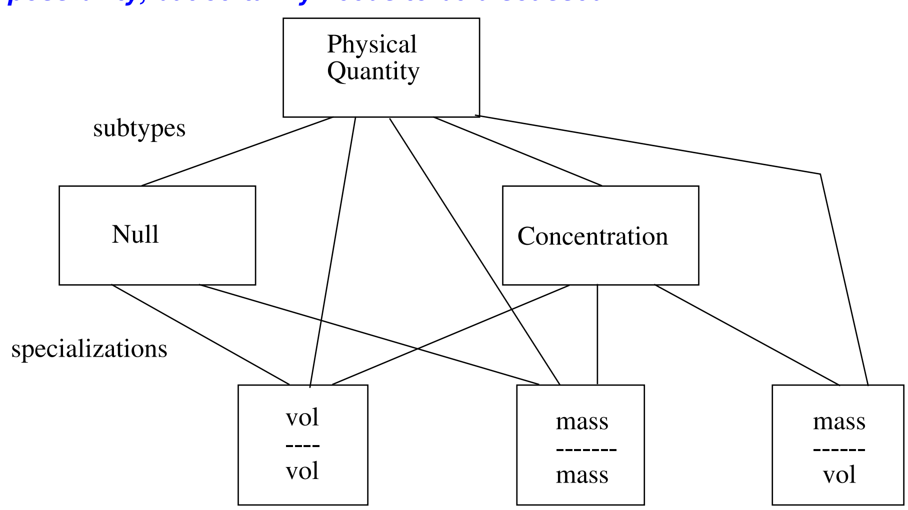
- Aggregates are ways of organizing elements of existing dimensions, but are not themselves dimensions. They don't add anything to the type graph, nor remove anything from it.
- Point "dimensions" are not physical quantities in the sense we defined in Section 34.7. ***What are they? Are they siblings of PQ?***
- Specialized and vector dimensions are themselves dimensions. One of them can be used anywhere that a dimension can: in an aggregate, as a member of a quasi-dimension, as a difference between two points, as an element of a vector, as the object or result of a specialization. ***What is the canonical form of a specialized dimension? of a vector dimension?***
- We added a new relation, "specialization" to the type graph.

>>ToDo 46: Stephanie's questions:

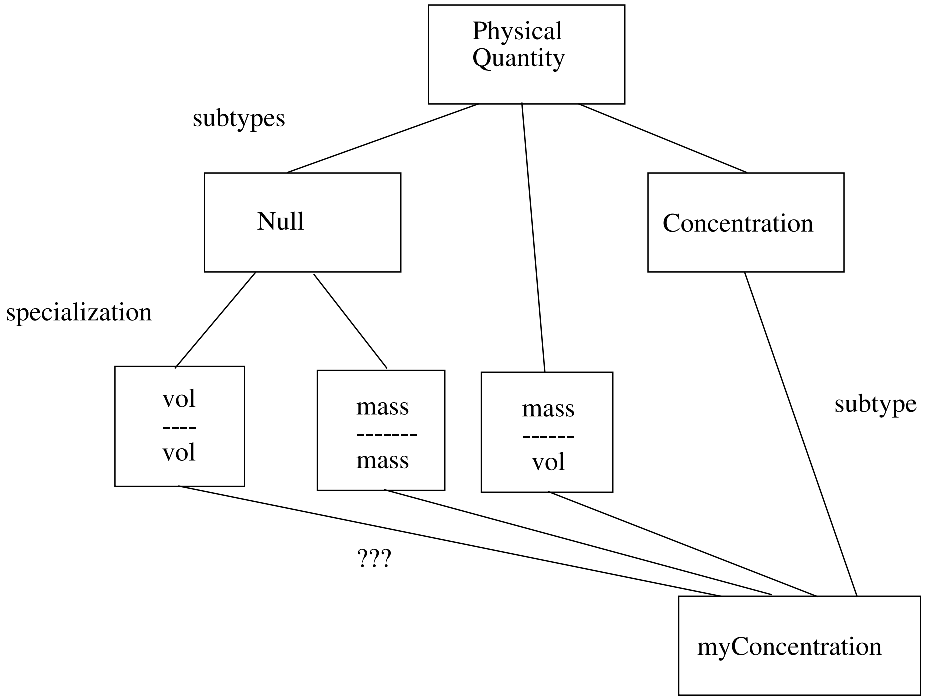
Question: We need to answer the questions regarding where generalized dimensions fit in the type hierarchy, whether or not quasi-dimensions can be formed from specialized dimensions (and vice-versa), and what is the most general form of a dimension.

I haven't addressed yet whether or not quasi-dimensions can be formed from specialized dimensions or vice-versa. I think that depends on how quasi-dimensions fit into the type hierarchy (if at all), so I will address that issue first.

Quasi-dimensions are not currently in the type hierarchy. This means that "time" can be a subtype of PQ, but "concentration" cannot. This also means that we can't specialize a quasi-dimension. This doesn't feel right, but allowing concentration to be a subtype of PQ leads to lots of messy hierarchies. Here are some alternatives we have looked at, each of which have quasi-dimensions in the type hierarchy, under PQ. The first two alternatives are not good (as described below). The last alternative might be a possibility, but certainly needs to be discussed.

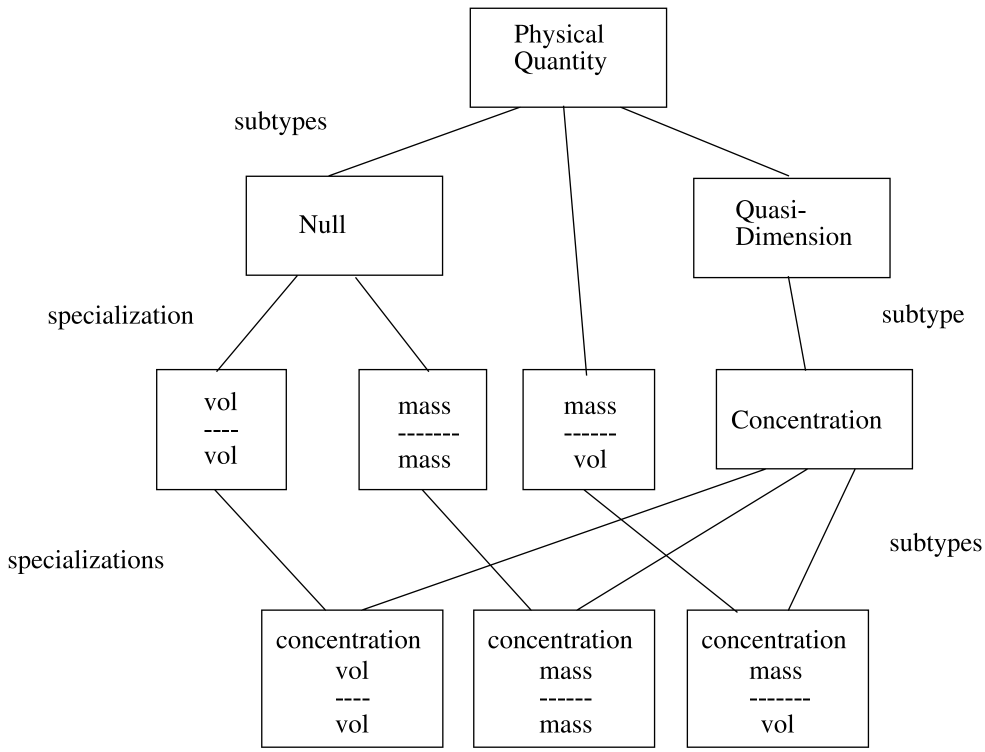


The biggest problems with this type hierarchy is that vol/vol and mass/mass are going to be specializations of both Null and Concentration, and mass/vol is always going to be a specialization of concentration, no matter how it is used.



The problem with this type hierarchy is that it is not clear what kind of relationship exists between myConcentration and vol/vol, mass/mass and mass/vol.

The following I think is a possible way to include things like Concentration in the PQ type hierarchy:



In this hierarchy, all subtypes of a concentration are quasi-compatible (it is possible to convert among them as discussed in the section on quasi-dimensions). In addition, each specialization making up the subtypes of a concentration has a unique canonical representation. Thus, Concentration is an abstract supertype, and all instances of it must be instances of one of its subtypes.

Question: There is a note that in order to do static type checking, we probably have to introduce dimensional subtypes of PQEs, so that we know that height(Bill) is being assigned to a length PQE and not a time PQE. Are we sure of this? What else do we need for static type checking?

I believe that the only types we will need are SPQE and VPQE, with PQE as an abstract supertype. We need SQPE to check for arguments to the functions

dimension(SPQE) -> ScalarDimension

units(SPQE) -> Unit

number(SPQE) -> Number

These are the only PQE types we need because all of the rest of the functions needing type checking would be on PQ subtypes, such as

height(Person) -> Length

rather than on SQPE subtypes. The automatic conversions between PQEs and PQs should make this kind of type checking possible. (This is assuming that we can determine the dimension of a PQE from its unit - the unique names argument. If this is not possible, we probably do have to have dimensional subtypes of PQEs, one for each dimension.)

One reason that we might want to subtype SPQE and VPQE is to support storage of a PQ as a specific kind of PQE (perhaps we might want to store all distances in meters). Another possible use might be to support the requirement of storing the exact string entered (although this might really require storing a string instead of a PQ). I can't come up with a compelling need to subtype SPQE and VPQE - these are just some areas where it seemed like it might be useful.

Question: can we always determine the type of an expression, even if it includes quasi and/or specialized dimensions?

First step - check for legality. If the expression is not legal, than we do not need to determine its type.

Does the expression have specialized dimensions in it?

No: use standard dimensional analysis to determine the type

Yes: is the resulting dimension named? For example, if we get the expression

$$\frac{specDim1}{specDim2}$$

and we know that

$$\frac{specDim1}{specDim2} = specDim3$$

then we would use specDim3 as the resulting dimension. If the resulting dimension has not been named, then treat the specialized dimensions as their associated canonical forms, and use standard dimensional analysis to determine the result.

Does the expression have quasi-dimensions in it?

I don't know how to handle this, For example, what should

$$\frac{concentration}{time}$$

be considered as? I think it might depend on how we put quasi-dimensions into the type hierarchy, but I'm not sure.

IV UNITS AND CONVERSION

17 Units Stuff

***>>ToDo 47:* This all needs review and reorganization.**

- How to recognize unit names.
- How to infer dimensions from units.
- Conversions: when, how, results.
- Defaults.

Some of this may overlap the language stuff later.

17.1 Units

To communicate the value of a physical quantity between a computing system and its clients, we use a thing very similar to the "unit" discussed in Section 3.4, but which need not be a physical quantity. We discuss these here.

Once we finish Section 34.7, clarify the above: the unit discussed above is, by definition, an amount of mass, length, etc.; whereas the unit discussed below is simply a mapping, and need not be any physical quantity at all.

17.1.1 Straightforward Units

A unit is a mapping from a dimension to a number. For example,

Kilograms(Mass) $\rightarrow$ Number.

A *simple unit* is one with no implied multiplication, such as feet, kilograms, or watts. A *compound unit* is a multiplicative combination of simple units, such as feet/second or watt-hours. Compound units are usually reduced to lowest terms, but not necessarily: units such as inches/inch or seconds/second are also legal, and occasionally useful.

Some derived dimensions have both simple and compound units associated with them; for example, voltage and resistance each have their own simple units. In such cases, the corresponding compound unit is rarely used (nobody records resistance as volts/amp). Other derived dimensions are expressed only in compound units, such as feet/second.

There are issues about how to infer the dimension of a quantity from its units on input, and how to express its dimension by means of units on output. We discuss those in Section 22, Section 34.13, and Section 29.3.8.

***>>ToDo 48:* Where do we talk about several dimensions having the same units?**

17.1.2 More Complex Units

The units mentioned in Section 17.1 are physically and mathematically well-behaved. We mention here some of the complexities which units can exhibit. Some of these complexities occur because the mappings are not counts of unit quantities; those are outside the scope of this discussion. Others are counts, but still exhibit complex behavior; these suggest richer functionality than is currently found in systems supporting dimensioned data.

Although most units map the 'nil' value of a physical quantity to zero (no mass is mapped to 0 kilograms, slugs, or ounces; no length is mapped to 0 miles, kilometers, or parsecs, etc.), this is not required. The Fahrenheit and Celsius temperature scales reach 0° at temperatures which physicists do not call "no temperature." The Gregorian calendar reaches 0 at a date which cosmologists (and even theologians) do not call the beginning of time. Logarithmic scales usually measure ratios, and reach a value of 0 for ratios of 1.

>>ToDo 49: Does a dimension have to be totally ordered?

We said above that a unit maps from a dimension to a number. More generally, a unit maps from a dimension to a subset of the real numbers, and still more generally, to any ordered set ***[doesn't it have to be totally ordered?]***. Examples of such sets are:

- Compass points (numeric, between 0 and 360)
- Compass points (letter set, such as NNE)
- Nonnegative integers
- Rationals between 0 and 100, inclusive
- The categories Low, Medium, High
- The letter grades A–F

The concept of generalized units becomes especially important when we consider "soft" dimensions, such as are found in the social sciences or business. Things like intelligence, quality, productivity, efficiency, reusability, risk, net worth, complexity, usability, etc., are often measured quantitatively. The units for such measurements are difficult to determine and are often controversial. This paper is not concerned with the meaning of such quantities—only with ways to operate on measurements of them. Of course arithmetic is not possible on non-numeric values, but assignment, comparison and unit conversion are still meaningful.

A unit's mapping may be time-dependent. For example, the meaning of the volt changed at the start of 1990, and the value in U.S. Dollars of a British Pound is different on different days. Other units (such as "1960 Dollars") can be defined which bind the time parameter, but time-dependent units still exist. Note that by choosing one currency (such as the U.S. Dollar or the ounce of gold) as an "anchor" it is possible to isolate the value of commodities from fluctuations in the anchor currency. This is just a notational convenience, however, as all other currencies are still time-dependent.

A unit's mapping may be inherently approximate. For example, a "pinch" is nominally 1/8 teaspoons, but it is absurd to talk about 1.2 pinches of salt. The same is true of Few/Not Many/ Many—the same physical quantity may be mapped to different values at different times, and vice-versa.

>>ToDo 50: Units questions:

How should the range of units be generalized? For example, a unit is currently a mapping from a PQ to a number. This could be more general. Consider such examples as grades and compass points. How do the units and number functions work? Should the name of the number function be changed to accommodate this?

The most general range of a unit is a set (enumeration). Must members of that set be ordered? This brings up the associated issue: must all dimensions be ordered?

17.1.3 Unit Names

Note that a unit is not the same thing as its *name*. The mapping that maps Bill's height to the number 6 has the names "foot" and/or "feet," but it may have other names as well, and it need not have any name. It is also possible for the same name to refer to any of several units, perhaps corresponding to different dimensions. For example, the name "pounds" might refer to a unit of force or a unit of money (and thus the name of the unit does not unambiguously determine a dimension), and the name "ton" might mean short ton, long ton, or metric ton, all of which are in the mass dimension (so the name determines a dimension unambiguously, but the mapping is ambiguous). In each case there is no ambiguity about the dimension, nor the mapping associated with the unit, only about which unit is denoted by the name. This name $\rightarrow$unit ambiguity can be resolved by requiring the use of unambiguous unit names, such as “money.pounds” and “weight.pounds”, or “short-ton”, “long-ton” and “metric-ton,” or by restricting the set of units allowable in a given context.

As discussed in Section 19.5.2, the prefix “kilo-” can denote multiplication by 10^3 or 2^{10}. Again, this illustrates a name $\rightarrow$unit ambiguity, rather than any confusion about the unit’s mapping or associated dimension.

In this paper we are concerned with units, rather than their names, and we assume that the names used for units are unambiguous.

V COMPUTATIONAL SYSTEM SUPPORT

>>ToDo 51: Total review needed.

[THE REST OF THE DOCUMENT REALLY NEEDS A GOOD REVIEW. I'VE JUST SORT OF PILED IT ALL UP BACK HERE.]

18 Introduction

>>ToDo 52: To be completed.

This may not be worth a full section. Main message: we sometimes describe alternatives, and their tradeoffs, leaving it to language and system designers to choose.

Maybe also provide an overview of subsequent sections here.

18.1 Types of System Appropriate for Delivering Support

[Is this note from Bruce?] Characterize them. Ideally, of course, everything would support PQs. In practice, only those likely to deal often with information about the world, and large enough to amortize the effort. For example, engineering calculators, test executives, database management systems, programming languages.

19 Syntactic Matters

[Note. I'm using "syntax" to denote things a user sees, and "implementation" to denote things he doesn't see.]

19.1 Physical Quantity Expressions (PQEs)

Dimensioned data represents physical quantities. The expressions "6 feet" and "2 yards" represent the same length, just as a certain social security number and a certain employee number represent the same person, and just as 2 (in base 10) and 10 (in base 2) represent the same number. It is useful to imagine a system in which the internal representation of lengths is hidden from users, much as the internal representation of an object identifier might be hidden in an object system, or as the internal representation of a number is immaterial to the semantics of a programming language. If the values of x and y are lengths, then it should be possible to compare them, add them, or multiply them by a constant, all without having to specify the units of measurement. Furthermore, it should be possible to add 2 inches to the length of something without specifying the units of that length. On the other hand, it is necessary to express lengths in forms such as "6 feet" or "2 yards" for input and output of data; it is also likely that most implementations will store the data in some such form.

This implies a type system which distinguishes between physical quantities and the expressions (such as "6 feet") used to represent them. Units link the two concepts, providing the mappings between physical quantities and the numbers which occur in their representations. For example, the length of a car is a physical quantity (a length), and a concept in its own right, even if we do not specify any units or a number associated with that length. In fact, until a unit is specified, a number cannot meaningfully be associated with a physical quantity.

In a sense, these are adaptations of two principles of object technology: identity and encapsulation. The identity principle says that things have identity independent of their properties. The encapsulation principle says that the internal representations of things is immaterial to their external semantics, i.e., their behavior under operations.

A construct such as "2 feet" or "5 miles per hour at 90 degrees" is a *physical quantity expression* (PQE), whose evaluation yields a physical quantity. PQEs are analogous to numeric expressions, whose evaluations yield numbers, and to character expressions, whose evaluations yield character strings. There is a difference, however. Numbers and character strings have direct external representations of their own, while physical quantities do not—they can only be represented externally by PQEs. We do not propose specific syntax here for PQEs. Instead, we use informal notations such as “2 feet” and (2 feet) in examples.

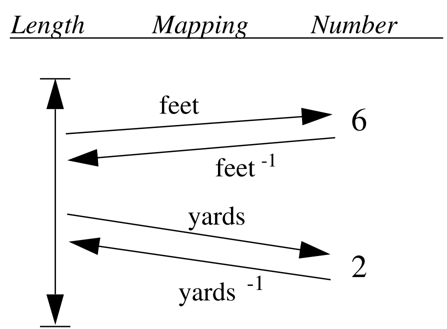
A PQE is not a physical quantity, but an expression which evaluates to a physical quantity. This convention simplifies the treatment of equality. The expression

(72 inches) = (6 feet)

is true in a natural way, since the two terms evaluate to the same physical quantity. The equality does not involve comparisons between 72 and 6 or between “inches” and “feet”.

Quoting conventions can be adopted for operations on the PQEs themselves. For example, it useful to have an operation which tests the units in which a PQE is expressed. Thus Unit(“72 inches”) returns “inch”, and does not refer to the underlying length itself. This is analogous to an operation which counts the terms in an arithmetic expression, e.g., Terms(“1+3”) is two, having nothing to do with the number four (which is the value of the expression). The Units operator helps reinforce the distinction between Physical Quantity Expressions and Physical Quantities. It makes sense to speak of the unit in “4.25 inches,” but it does not make sense to speak of the unit in the width of my hand.

Another way to see the appropriateness of treating PQEs as expressions is to see them as a syntactic variant of traditional expressions. PQEs such as “2 feet”, “5 m.p.h.” or “16 kilograms” are essentially just peculiar syntax for the functional inverse of the mappings “feet,” “m.p.h,” and “kilograms.” Suppose, for example, that the value of x is a certain length, denoted by |↔|. Then the unit mapping feet(x) might yield 6, signifying a length of 6 feet. The expression feet^{-1}(6) is the inverse,[1] mapping back from 6 to |↔|. Thus the PQE (6 feet) can be read as feet^{-1}(6); its value is that length which is mapped by feet into 6.



Just as physical quantities can be broken down into scalar and vector physical quantities, a PQE can be either a *scalar physical quantity expression* (SPQE) or a *vector physical quantity expression* (VPQE). VPQEs are described in Section 19.4.

Every SPQE has an associated number, unit, and dimension. SPQEs have the following functions defined on them:

1. The functional inverse, not the multiplicative inverse. I.e., a function which maps numbers of feet to Length, not 1/feet.

dimension(SPQE) $\rightarrow$ Scalar Dimension

units(SPQE) $\rightarrow$ Unit

number(SPQE) $\rightarrow$ Number

***>>ToDo 53:* SPQE's in the type graph?**

[Observe that such function signatures imply that we have SPQE in the type graph. It would be interesting to generalize that to arbitrary expressions as objects.]

Stephanie's question: Our description of functions defined on SPQEs implies that SPQE is in the type hierarchy. This is mentioned, but not pursued.

My first inclination was not to put PQE (or SPQE) into the type graph. After all, integers are in the type graph, but there is no equivalent "integer expression" in the type hierarchy. Since our rules for conversion between PQs and PQEs, such as

height(Person) -> length

allows statements such as

height(Bill) := 6 feet

we seem not to need an explicit PQE type.

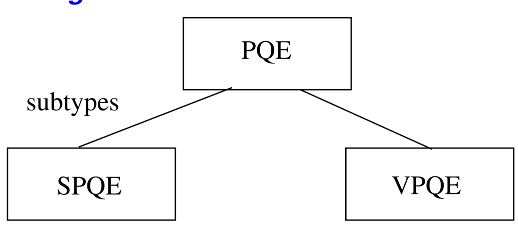
However, if we want to support the functions

dimension(SPQE) -> ScalarDimension

units(SPQE) -> Unit

number(SPQE) -> Number

then we pretty much have to have SPQE in the type graph, since neither the units nor the number function makes sense on a PQ. In this one case, we do distinguish between PQs and PQEs. So, my recommendation is to put PQE, SPQE (and perhaps VPQE, depending on how we handle vector dimensions) into the type hierarchy. I would imagine the following:



It should be assumed that the arguments are quoted, i.e., not evaluated to the underlying physical quantities. If units uniquely determine dimensions (which may not be the case under dimension specialization [Section 23]), then the dimension component can be inferred from the unit.

19.2 Additive PQEs

We call an expression such as "5 feet, 5 inches" (used to represent a length of 65 inches) an *Additive PQE* (APQE). In an APQE, all elements belong to the same dimension, and are used to represent a single physical quantity, in much the same way that an expression such as 1+2 represents the single number 3. APQEs are useful in many circumstances, such as for representing time (hours, minutes) or date (year, month, day). All of these examples have the property that they divide the representation of a single physical quantity into units of different granularity in a (coarse, fine, finer, finest) format, with the result implicitly the sum of the elements of the APQE. For example, feet is a coarser measurement of length than is inches, and “5 feet, 5 inches” represents a single length (65 inches) that is the sum of the elements.

The elements of an APQE are ordinary PQEs, not vector PQEs or other APQEs. The Dimension() function of an APQE is the same as for an ordinary PQE. Unit() and Number() functions are not directly meaningful, but if we provide a way of decomposing an APQE into its constituent PQEs, then Unit() and Number() functions can be applied to these. Except for this fact andm certain I/O issues , an APQE is treated just like the corresponding sum.

In contrast to a VPQE, an APQE representing a specific physical quantity does not have a fixed number of elements. For example, an APQE representing 2.5 meters might be written as “2 meters, 50 centimeters,” or “2 meters, 40 centimeters, 100 millimeters,” or even “2 meters, 19 inches, 1 centimeter, 7.4 millimeters.”

The representation of APQEs is discussed in Section 29.3.5.

>>ToDo 54:* *More needed?

Stephanie’s question: What do we want to say about APQEs? Are we satisfied with the level of detail we have? Do we want to address how to distinguish a VPQE from an APQE? Do we care where APQEs are in the type hierarchy?

We say that APQEs do not need to go into the hierarchy, as they are constructed only as necessary. Are we sure that this means they don’t need a place in the type hierarchy? How can we convert from an APQE to a PQ if we don’t have functions (presumably on APQE) available to do the conversion?

19.3 PQEs With Non-numeric Units

When physical quantities are mapped into non-numeric values (A/B/C/D/F, pass/fail, high/medium/low, North/East/South/West), common usage omits the name of the unit, and the resulting expression is just the corresponding value. We know how to allow administrators to define such units and how to store them (as user-written functions), and we know how to represent their outputs (as ordered sets with string-valued print representations); but we do not know how to tell on input which dimension is intended (since we can’t force an unambiguous unit name). Conversion among such units is difficult, and is discussed in Section 34.11.

19.4 Vector Physical Quantity Expressions

A *vector physical quantity expression* (VPQE) represents a vector physical quantity, such as velocity. Each component of a VPQE is a scalar physical quantity expression (SPQE), with an associated dimension, unit and number. Access to the individual components of a VPQE can be provided in at least two ways. They can be accessed by position number (in which case the ordering of the components is significant), or the components can be individually named, and thus accessed by name. In either case, there must be a mechanism for determining the number of components of a vector. If components are accessed by name, there must be a mechanism for determining the names of all the components. A coordinate system is associated with a VPQE; it is outside the scope of this discussion how this information is communicated between a system and its users. A dimension is also associated with the VPQE itself, separately from the dimensions of its components. Two vectors with a different associated dimension are unequal (actually, incommensurate), even if they are element-by-element equal.

Recall that a scalar physical quantity expression (SPQE) has the form:

(number unit)

A VPQE has the form:

<SPQE, SPQE, …>

For example, a VPQE representing a velocity might look like

<(5 m.p.h), (170 degrees)>

In defining a new vector dimension, a user needs to provide:

- The name of the dimension (must be unique)
- The dimension of each component
- The mappings (in both directions) between each of the permissible coordinate systems and the *reference* coordinate system (the reference coordinate system is discussed below). For the convenience of users and administrators, the coordinate systems will probably also have names.

19.5 Units Recognition

>>ToDo 55: To be completed.

Topics:

- Composite units: feet/second/second.
- Homonyms: pounds weight vs. pounds money.
- Synonyms: pound vs. pounds vs. lb vs. lbs. Miles/hour vs. mph.
- Implicit units from declarations, defaults.
- Prefixes.
- Inferring dimensions from units.

19.5.1 Derived Unit Names

>>ToDo 56: To be completed. Are we agreed on this stuff?

19.5.2 Prefixes

It is possible to recognize any of the SI prefixes [1] in front of an SI unit, and generate the appropriate multiple. Thus, once grams are known it would not be necessary to define micrograms. [1] recommends that the SI prefixes not be applied to other than SI units, so that a system would not automatically recognize nor generate “femto-yards”. Note, however, that the standard also recommends that “yards” not be used at all. If a system chooses to follow this recommendation, that implies that prefixes are not automatically applied to user-defined units—if a user wants to use kilodollars, they must be defined explicitly. [1] gives other practical rules for use of the prefixes.

The prefix “kilo-” can denote multiplication by 10^3 or 2^{10}, and similarly for mega-, giga-, etc., but not for milli-, micro-, etc. This is discussed in [23]—generally, kilo- means 2^{10} for addressable computer storage, and it means 10^3 for everything else. Users need a way to tell a system which is intended, or the system needs a way to infer it. Either of those is outside the scope of this paper.

A system designer has the choice to allow the set of prefixes to be extended. On the one hand, the set changes very slowly. On the other hand, certain professions or countries may wish to add prefixes.

19.6 Coordinate Systems

>>ToDo 57: Why is this here?

[Relate to my observation somewhere else that the units themselves might incorporate a notion of "origin".]

PQEs are sometimes understood with respect to some coordinate system. Sometimes the PQE is embedded in a statement which names the coordinate system explicitly, as in "5 miles west of Cleveland," and sometimes the coordinate system is understood from the context ("he gained 5 yards") or from the choice of unit ("250 ° Celsius," "37.5 degrees West, 12.1 degrees North"). The choice of coordinate system affects the meaning of addition of PQEs, multiplication of SPQEs by a dimensionless quantity, and multiplication of VPQEs by a dimensionless scalar.

If enough information is known, quantities can be converted between coordinate systems, in much the same way as they can be converted between units. Support for dimensioned data can include means for associating a coordinate system with a PQE, for converting between coordinate systems as needed, and for adding, changing and deleting coordinate systems. See also Section 17.1.2, Section 34.7, and Section 12.

20 Basic Computational Support

Basic dimensional support deals with the case in which there is a 1:1 correspondence between dimensions and their associated canonical forms. If a problem is well-modeled by (or can be restricted to) dimensions which satisfy this property, then what we describe as basic dimensional support applies to that problem.

20.1 Declaring Dimensioned Data

A system supporting dimensioned data should provide a type hierarchy, populated with an initial set of dimensions and units. The SI reference units [1, 4] are a good starting point, possibly augmented by money. Weight can be provided in addition to Mass (or instead of it). The ability to introduce new dimensions and units is described in Section 21.1.

Such a system also provides capabilities to declare the types of variables and function arguments and results.

>>ToDo 58: Declaring variables with and without units.

[Discuss or refer to the tradeoffs between declaring with and without units. Accommodate both.]

Attribute functions (e.g., the height function defined on persons) and variables can be declared to have a dimensional type such as Length. Thus the signature of the height function might take the form

height(Person) $\rightarrow$ Length.

Units need not be associated directly with attributes and variables of dimensional type, in the same way that units are not associated with instances of physical quantities. Different units may be used when assigning or retrieving the values of an attribute or variable at different times.

One could associate units with such variables, if one wanted. For example, we might declare the variable myHeight to be a length in feet. In this case, the only values it could contain would be lengths measured in feet. Variables like this are useful in expressions like "heart_rate ≤ age * 1.5," where heart_rate is in beats/minute and age is in years.[1] The desired coercion can be stated once in the declaration, and subsequent use is clearer and less error-prone.

---

1. It is arguable that this is an expression among numbers, rather than among physical quantities. We take no stand on that issue—if users want to treat such quantities as numbers, existing systems will do that; if they want to treat them as physical quantities, then this paper explains how.

For most purposes such variables (or functions) behave just like any other variable of dimensional type. The usual defaults and coercions would still apply on input and output. A few differences might be visible:

- If the variable's numeric representation had low precision (if it held only integers, for example), assignment to it would introduce visible rounding or truncation. Even if all variables have low precision, so that visible rounding and truncation are the norm, associating a particular unit with a variable may cause the rounding or truncation to occur for different values, or in a different direction.
- Having the units implicit might make it possible for the system to conserve storage. On the other hand, storage defaults could make *all* units implicit, in which case no difference would be visible.
- Depending upon where such a variable resided in the type hierarchy, it might be compatible with the type *number*. In that case expressions like "myHeight+3" would be legal.

>>ToDo 59: More implementation considerations.

Possibly make statements about storage. Assignment, equality, arithmetic obey compatibility rules. Rules for input/output already described in section on PQEs. Coercion to/from non-PQ types, especially from literals.

20.2 Operating with Dimensioned Data

Operations on dimensional data are governed by the type constraints on the dimensions involved. Units participate indirectly; they identify the dimensions involved, and they may cause some conversions to be applied during computations.

20.2.1 Compatibility

Common operations behave differently for dimensioned data. For example, while "2+5" is a legal expression, "2 miles + 5 hours" is not. These changes affect operations such as assignment, comparison, and arithmetic. We describe only binary operations here, but the concepts can be applied to unary or n-ary operations as well. The following questions arise:

- Which operations should be considered?
- Which combinations of dimensioned and undimensioned operands are legal for each operator?
- What is the dimension of the result?
- How is the operation to be performed? The answer may involve:
  - Ordinary operations on the numeric components.
  - Implicit units conversion [Section 20.2.3].
  - Special operations (e.g., on vector dimensions [Section 26.3]).

There is a spectrum of possible approaches to these questions:

1. Simple, fixed rules.
2. Rules dependent on particular combinations of dimensions and operations.
3. More sophisticated rules taking other context into account.
4. Modifiable rules which can be extended and tailored by users:
   - Editable rules.
   - Special operations to override the rules, e.g., casting or coercion functions.

We estimate that level 1 is adequate for basic dimensional support, or level 2, if one is working with quantities like temperature [Section 6.2].

Basic dimensional support is based on these fixed, simple rules:

- Assignment and comparison are valid between terms having equivalent dimensions.
- Addition and subtraction are valid between terms having equivalent dimensions. The resulting dimension is the same.
- Multiplication and division of any dimensioned term by any dimensioned or undimensioned term is legal. The dimension of the result is determined by algebraic reduction of the canonical forms of the dimensions of the terms, possibly yielding a derived dimension.

Simple rules such as these are inadequate in several ways:

- They do not account for all operations, e.g., exponentiation, absolute value, vector operations, etc.
- Equivalence of dimensions is not always well-defined.
- The generalizations are not always valid. It may not be meaningful to add two dates, or two temperature points, nor to multiply a date or temperature point by a constant.
- There may be circumstances requiring rules to be relaxed, as in the computation of certain metrics. For example, a shipping company might accept only parcels for which the length + girth in inches, plus the weight in pounds, is less than 130.

More elaborate rules might take the form of a matrix of dimensions and operators, such as:

D1	D2	Operators					
		+	-	*	/	assignment	comparison

D1 and D2 stand for the dimensions of the operands; one operand might be undimensioned. The matrix might have additional operators as necessary, such as < and >, etc. Each cell in the matrix would contain the following information:

- the legality of the operation
- special operations that might have to be performed (if the operation is legal)
- the dimension of the result (if the operation is legal)

It may be desirable for such a matrix to be editable by users, particularly for newly defined dimensions. Addition of new operators is also desirable.

>>ToDo 60: Need more work on matrixes.

Further work is needed to determine whether simplifications of this matrix form would be useful. The role of this matrix should be expanded upon.

When dealing with both arithmetic operators and assignment, the legality of a statement can be determined by the following steps:

- Can the expression on the right hand side of the assignment be formed? That is, are the dimensions compatible for the operators used?

- If so, is the resulting expression compatible with the destination on the left hand side of the assignment?

For example, the legality of Mass ← Length + Length is checked as follows:

- Length + Length is legal, and gives a Length.
- Mass ← Length is illegal.
- Therefore, the operation is illegal.

Notice that this matrix may need to be extended to handle generalized units. For example, 360° = 0° in some angular measures.

20.2.2 Mapping Between Physical Quantities and Physical Quantity Expressions

>>ToDo 61: Needs work.

Revisit this section, answering questions where possible. Defer representation issues to Section 29.3.

Simple Units

Mappings between physical quantities (i.e., their internal representations) and physical quantity expressions can be obtained by explicit functions. Defaulting mechanisms to simplify the language constructs are described in Section 22.

A dimensioned attribute, such as the height function defined on persons, returns a physical quantity in some unspecified internal representation. For illustrative purposes, we might write

height(Bill) = |↔|

where we use |↔| to denote a length. The value may be stored internally as a PQE in some units, but that is a matter of implementation. The semantics of the height function is that it returns a length, not a PQE.

There are no inherent units associated with a length. The result of height(Bill) can be compared with other lengths, multiplied by a constant, and otherwise manipulated without specifying anything about units. For example, one can add 6 inches to it without specifying the conversion between the units in which height(Bill) was entered/stored and inches.

However, this result cannot be externalized, i.e., displayed or exported, without conversion to an SPQE such as “6 feet”. For this purpose, functions are needed for mapping a length into an SPQE. They can be provided as a family of functions, one for each unit, e.g.,

feetSPQE(Length) → SPQE,

inchSPQE(Length) → SPQE,

…

or as a single generalized function

measure(PhysicalQuantity, Unit) → SPQE.

A request to display the height of Dan in feet might then be expressed in one of the following forms:

display(feetSPQE(height(Dan)))

display(measure(height(Dan),Feet))

The number function [Section 19.1] could be used to display the numeric portion alone, as in

display(number(feetSPQE(height(Dan)))).

The data type used to represent the numeric part of an SPQE can be specified in the same way that the units are specified: by expanding the family of functions or through additional parameters to the "measure" function. Though considerable sophistication is possible (see Section 17.1.2), for many purposes it is sufficient simply to name one of the system's underlying numeric types, e.g., integer, double, etc. The numeric type for display is independent of the numeric type for storage. In general, both need to be specified, though a system of defaults can relieve most of the work most of the time (see Section 22). We omit the numeric type specification in the rest of this paper.

>>ToDo 62: Numeric representations.

Are we really convinced that the numeric representation can be handled so simply? Stephanie is not.

Defaulting conventions [Section 22] could be used to establish feet as the default unit for "casting" a length into an SPQE, so that height(Stephanie) is expressed in feet by default in an output context, such as display. A request to display the height of Stephanie might then take the simpler form

display(height(Stephanie)),

with the understanding that it will be mapped into an SPQE in feet.

[We could define measure(x,unit) as concat ("(", unit(x), unit, ")").]

Other Units

As illustrated above, mappings to PQEs can be provided as a family of functions, one for each unit, or as a single "measure" function taking the unit as argument.

When the units are not simple, the family of functions explodes combinatorially with the various combinations of simple units involved. Thus, for Speed, there would be functions for

feet_per_secondSPQE(Speed) $\rightarrow$ SPQE,

inch_per_secondSPQE(Speed) $\rightarrow$ SPQE,

mile_per_secondSPQE(Speed) $\rightarrow$ SPQE,

…

feet_per_hourSPQE(Speed) $\rightarrow$ SPQE,

inch_per_hourSPQE(Speed) $\rightarrow$ SPQE,

mile_per_hourSPQE(Speed) $\rightarrow$ SPQE,

…

The number of functions is even larger for vector dimensions, being the product of the number of functions available for each component. Thus the family of functions for velocity would be of the form

feet_per_second_compasspointVPQE(Velocity) $\rightarrow$ VPQE,

feet_per_second_degreesVPQE(Velocity) $\rightarrow$ VPQE,

…

It would be simpler to use a single "measure" function rather than a family of functions. However, this requires the ability to specify compound and vector units as arguments.

Whenever a unit name is constructed by implied multiplication or division ***[Can we construct them any other way?]***, there is risk of duplicating a predefined unit name, as in "foot-candle." This is an instance of the problem of ambiguous unit names, and is not in the scope of this paper.

Inverse Mappings

In principle, inverse mappings from PQEs to physical quantities are also needed. Strictly speaking, an assignment of the form

height(Bill) $\leftarrow$ (6 feet)

cannot be carried out directly, since "6 feet" is an SPQE, while height(Bill) requires a length. The precise expression would be of the form

height(Bill) $\leftarrow$ feetSPQE $^{-1}$(6 feet).

However, the treatment of PQEs as expressions [Section 19.1] eliminates the need for such explicit inverses. In the absence of a quoting context, the SPQE "6 feet" is treated as an expression which evaluates to a length, so that the assignment can in fact be legally written as

height(Bill) $\leftarrow$ (6 feet).

>>ToDo 63: A type checking question.

In order to do static type checking, we probably have to introduce dimensional subtypes of PQEs, so that we know that height(Bill) is being assigned a Length PQE and not a Time PQE.

Substitutability

The conventions described above imply:

- A PQE may occur wherever a physical quantity of the same dimension is expected, since the PQE will be evaluated to such a physical quantity.
- A physical quantity may occur wherever a PQE of the same dimension is expected if a suitable default mapping is in effect. Alternatively, the physical quantity may be wrapped in an explicit mapping function.

20.2.3 Unit Conversions

Conversions can be done automatically as needed between compatible PQEs when evaluating expressions. Not only can the conversions between units be automatic, but the internal units used for the calculation of the expression need not be constrained to be the units of the PQEs in the expression, nor to the units of the result. For example, it is possible to calculate the sum of two PQEs with units of feet, and produce a result in feet, but to do the actual addition in meters. In Section 29.3.1 we discuss the effects of different choices of unit for calculations and comparisons.

The conversion factors between units can be time-dependent or asymmetrical, as in currency conversions. Common usage employs the conversion rate in effect at the time each conversion is done. We know of no standard way to request other rates. [1] The measure of some quantities like uranium or savings accounts can change with time, but this is because of changes in the quantity, not because of a time-varying unit mapping.

Sometimes the prevailing set of conversions in a particular application is not consistent with the "true" conversion. For example, rent is often calculated on the basis of a 30-day month and a 12-month year, yielding a 360-day year. This anomaly, however, is simply confusion among different units with the same name (conventional months vs. 30-day months, etc.). Similarly, rent in February is generally the same amount as rent in March, although the months have different numbers of days. This gives a varying per-day rent, depending on the length of the month. The anomaly here is simply that the per-day rent varies with the month, however, not that there is anything unusual about the units involved.

1. Though a syntax could be invented, similar to "1960 dollars ."

We discuss the representation of units in Section 29.3.1. Conversion among non-numeric units is difficult. It is discussed in Section 34.11.

20.3 Input/Output

20.3.1 Input of Physical Quantity Expressions (PQEs)

Users can present PQEs using any units that a system knows about. The input system must know:

- what to do if the unit is missing or ambiguous
- what to do if the number is missing
- how much precision is intended in the number

Systems may also be expected to “know” about units with SI prefixes [Section 19.5.2]. In addition, systems may wish to allow implied multiplicative combinations of known units. For example, if a system knows about “feet” and “second” but not “feet/second,” users may still expect the system to understand “6 feet/second,” and infer the appropriate dimension (Length/Time) and unit (foot/second). In a similar way, the PQE “0.3 watt-hours” implies multiplication. Known units take precedence over implied multiplication: 1 foot-candle is not 1 foot $\times$ 1 candle.

>>ToDo 64: Dan, can you suggest a better example?

An Additive PQE can be hard to tell from a Vector PQE. Users may be required to use special punctuation for one or the other, so that a system can tell them apart. Once it is known which is being presented, each can be treated as the appropriate collection of SPQEs.

There is a loose notion of well-formedness for APQEs. Let the elements be $e_1, e_2, \ldots e_n$, $n>1$, and their units be $u_1 \ldots u_n$. Then,

- All elements belong to the same dimension.
- For $1 \le j<n$, Number(e_j) is an integer and is ≥ 1. This forbids “2.5 years, 7 months” “2 years, -4 months, 6 days.” It also forbids “0 years, 0 months, 6 days” and “2 years, 0 months, 6 days,” which might be legal in some contexts.
- Number(e_n) > 0. This allows the final element to be a non-integer, but still requires it to be strictly positive.
- For $1 \le j<n$, $e_{j+1} < u_j^{-1}(1)$. I.e., each quantity should be <1 of the unit on its left (remember, $\text{feet}^{-1}(3)$ is the length denoted by “3 feet.”) This forbids “2 years, 12 months, 6 days.”
- For $1 \le j<n$, $u_{j+1}^{-1}(1) < u_j^{-1}(1)$. This is the “coarse, fine, finer” rule. Note that it forbids “2 years, 2 years, 4 months, 6 days.”

A system may decide to check these rules on input and obey them on output. Note that there is no way to express a negative quantity as an APQE obeying these rules. Systems will need a syntax for expressing the additive inverse of an entire expression, and it will have to distinguish “<-6 feet, 3 inches>” (which is probably an error) from “-<6 feet, 3 inches>.”

20.3.2 Output of Physical Quantity Expressions (PQEs)

There are in general several ways to output a physical quantity as a PQE. Several dimensions may be candidates—this is discussed under Specialized Dimensions, in Section 23.4. Given a dimension, any of the associated units is a candidate, and the choice may be influenced by defaults and by human factors. For example, “ohms” may be preferred to “volts/ampere,” or “watt-hours” to “newton-meters.” This is discussed in Section 21.1 and Section 22.

If a quantity is to be expressed as an Additive PQE, then one needs to be constructed, possibly following the rules given in the previous section. Even obeying those restrictions, however there is more than one way to express a given physical quantity as an APQE. This can be an issue if repeatability across implementations is desired.

A system to support output of APQEs should let users (or administrators) indicate coarsest and finest applicable units, signal when values are to be presented as APQEs, and possibly specify families of units (to avoid expressions like "1 liter, 1 quart, 2 pints, 3 ml").

21 Extensibility

21.1 User-Defined Dimensions

While a system to support dimensioned data would doubtless be preconfigured to support some set of dimensions (perhaps the SI set, plus money), it could also allow the set of dimensions to be extended. It might even allow the preconfigured dimensions to be overridden or deleted. ***Note: money is the dimension; the various currencies are its units.*** A logical base configuration would be the SI set of dimensions, together with their base units. A system might also have the "money" dimension preconfigured, with the local currency as its base unit.

The definition of a dimension requires:

- One or more names.
- Status as reference or derived.
- If derived, the dimension(s) it is derived from and the type of derivation (multiplicative combination, specialization, vector).
- A unit. Additional units for this dimension will be defined in terms of their conversion to and from this unit. It will also be used for storage and I/O until default units are specified.

  A system might also give a derived dimension all the compound units implied by its canonical form. For example, Acceleration might have all the units implied by Length/Time^2, such as miles/hour^2, feet/sec^2, etc.

Note that it is not necessary to define derived dimensions such as Speed, since basic dimensional support already addresses multiplicative combinations of reference dimensions. It may be useful, though, to associate particular names or units with a frequently-used combination. In some contexts, for example, it might be convenient to define Acceleration, and specify "gravity" as one of its units.

A further level of support would allow users to define aggregates, quasi-dimensions, and point/interval pairs.

21.2 User-Defined Units

One might also like to add new units, associated with existing dimensions. The definition of a unit requires:

- The associated dimension. Note that this must already be defined.
- One or more names. Although unit names must be unambiguous (see Section 17.1.3), there is no harm in having multiple names for a single unit. ***Mention I/O issues in Section 5.***
- Conversions to and from other units (see Section 20.2.3). Depending upon the way the system is organized and the freedom it permits, it may be possible to specify the conversion in only one direction, or conversion to/from only the reference unit. Note that the machinery for adding a unit needs to be expressive enough to define the conversion between simple and compound units.

Aliases for existing units (such as "fps" for "feet/second") may be specified as such, or defined separately, with a conversion factor of 1.0. A system might be preconfigured with the SI special names for derived units, such as the *joule*, *hertz*, etc. [1]

22 Default Units and Data Types

There are a number of places where defaults would ease the burden of having to say something like "measurement-in-feet-displayed-as-an-integer". In addition, users may wish to have a system act as though amounts of money were always stored in U.S. dollars. A system could set defaults separately for input, output, and storage, and defaults could cover units and numeric format.

22.1 Units

Units can be defaulted in the following places:

Default Level	Input	Output	Storage
Dimension			
Function or variable			
Function/argument pair			
User Session			
Manual Override			

The table is in order of increasing precedence. For example, if we had the following input defaults in the system,

default for Length is feet

default for function height is inches

default for Stephanie as argument of function height is cm

default for Bruce's session is meters

then we would get the following defaults:

Function Call	User	Units
length(myCar) ← 10	Bill	feet
height(Bill) ← 72	Stephanie	inches
height(Stephanie) ← 150	Bill	cm
height(Bruce) ← 1.8	Bruce	meters
height(Dan) ← 1.9 yards	Bill	yards
height(John) ← 1.7 yards	Bruce	yards

Two additional levels of defaults could be

- system-defined by dimension, which would be lower precedence than the user-defined by dimension (described above)
- by argument object, regardless of the function in which the argument appears. For example, all the measurements taken in Dan's house might be in feet, but all measurements taken in Bruce's house might be in meters.

There are even more possibilities. For example, "as supplied" is a meaningful default for output and storage units. Similarly, "as stored" is a meaningful default for output. This is not meant to be a complete list of the default options, but it gives the flavor of what could be provided.

***>>ToDo 65:* A question.**

[SL:] This discussion of default units was written with reference dimensions in mind. Does this mean that we are proposing to let derived and vector dimensions "inherit" their defaults from the defaults of the reference dimensions? I think this would be a bad idea - for one thing, this wouldn't let us say that ohms should be used instead of volts/amp. We should extend this default system to derived and vector dimensions, so that defaults can be specified for ALL dimensions. (Maybe reference defaults can serve as "default" defaults.)

22.2 Numeric Format

Just as one can define defaults for the units part of a measurement, one can define defaults for the number part of a measurement. For example, we might want all lengths to be displayed in scientific notation, and all prices truncated to two places after the decimal place. Such defaults could use the same precedence rules and framework as described above. A remaining issue is how to provide such syntax.

22.3 Numeric Type

One could also specify defaults for the numeric type into which the numeric part of measured quantities will be stored. However, this is semantically separate from defaults for units and numeric format (I/O), as storage is inherently implementation-dependent.

23 Specialized Dimensions

It was noted in Section 8.2 that specialized dimensions behave like what the relational algebra calls "domains" [12, 14]. This behavior need not be implemented in SQL ***[<Reference>]***, nor by anything called "domain." For an implementation in C++, see Section 34.10.

23.1 Defining Specialized Dimensions

The definition of a specialized dimension includes:

- Its name
- The dimension being specialized. This can be a reference dimension, a multiplicative combination of reference dimensions, or a previously-defined specialization.
- The unit(s) associated with this specialization (which may simply be all the parent's units)

There is a tradeoff in the association of units with specializations. On the one hand, if units are uniquely associated with specializations (for example, if "foot-pounds" is associated with Torque but not with Energy or Force $\times$ Length), then it is possible to infer the intended specialization from the user's choice of units on input, and to express it through the choice of units on output. On the other hand, if a unit can be associated with more than one specialization, or with a specialization and its parent, then users gain flexibility. For example, it is convenient to be able to measure both (general)

Length and (specific) Width in inches, or all ratios of the form D_i/D_i in "percent." We know of no way to choose one half of this tradeoff over the other.

One question arises if a unit can be associated with more than one specialization, or with a specialization and its parent: we stated in Section 17.1.1 that units are uniquely associated with dimensions.

>>ToDo 66: Questions about units.

Do we need to change that rule here? If we keep it,

-the effect is as if we made a copy of the unit for each specialization, with the same name and conversion functions, but with a different identity and dimensional association. We can perform compatibility-checking, conversion, I/O, etc., but we have two different units with the same name. This can make administration difficult.

If we break it,

-then a unit can be associated with more than one dimension (hopefully, all of them members of a tree rooted at the same parent dimension). We need to know the dimension of a PQE for compatibility-checking, but logically we are allowed to know that directly, without reference to its unit.

-The original meaning of unit as a physical instance is lost if something is required simultaneously to be a torque and an energy. We don't need that here, but it's physically unsatisfying.

I can't see any other difficulties.

23.2 Declaring Specialized Dimensions

>>ToDo 67: To be completed.

How to declare a variable to be of a particular specialization, and how to interrogate the specialization of an already-declared one (in fact, we should be able to interrogate its whole path up/down the type tree, so that we can correctly label variables on output).

23.3 Operating With Specialized Dimensions

>>ToDo 68: To be completed.

23.4 Input/Output

Just as with ordinary dimensions, specialized dimensions are represented on input and output by PQEs. However, it can be ambiguous which specialization is being represented. This is an issue on both input and output. If, as mentioned above, the applicable units are uniquely associated with a specialization (or with the parent), then the choice of unit resolves the ambiguity. If not, then the specialization must be supplied explicitly. We do not propose any particular syntax for this.

>>ToDo 69: What is the type of "3 feet"?

What is the type of "3 feet"? Several possibilities have been suggested:

- It is strictly of type Length. Explicit casting is required to use it as a specialization:

  xWidth ← L2W(3 feet)
  or
  xWidth ← MakeWidth(3 feet)

- It is recognized as belonging to the “family” of Length and its specializations, being compatible with any of them.

Another point. We said that specializations inherit units, hence the unit set for Speed includes all pairs of the form Length-unit/Time-unit (in addition to any unique units of its own, like mph). Therefore the question we asked about “3 feet” also applies to “3 feet/10 seconds”. Is that compatible with Speed? Is the answer different for “0.3 feet/second”?

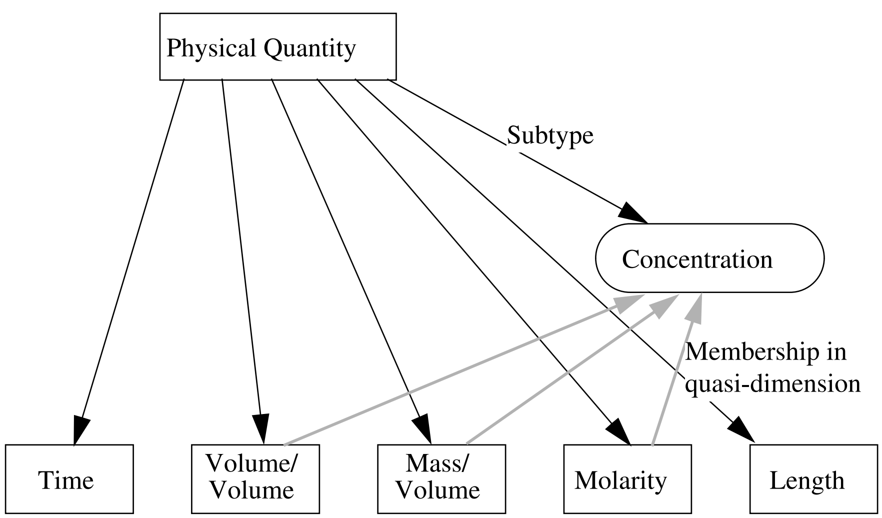
24 Quasi-Dimensions

24.1 Defining Quasi-Dimensions

The definition of a quasi-dimension includes:

- Its name
- The set of dimensions which it spans
- The rules for converting quantities among those dimensions

Since it corresponds to no single canonical form, a quasi-dimension is not itself a dimension. ***[Give a better reason.]*** Its name could be the name of an ordinary dimension without ambiguity, although users might find this confusing. No units are associated with a quasi-dimension. It is a subtype of Physical Quantity, but its children are related to it by ***[what?]***, rather than by subtype relations.



The dimensions spanned by a quasi-dimension retain their places in the type hierarchy, their units, etc. A dimension may be the child of more than one quasi-dimension, and quasi-dimensions may themselves be grouped into higher quasi-dimensions. It is worth noting that specialized dimensions are defined by their relation to their parents, while quasi-dimensions are defined by their relation to their children.

24.2 Declaring Quasi-Dimensions

>>ToDo 70: To be completed.

How to say which QDim(s) a variable belongs to, and how to interrogate that same information. Is QDim membership static or dynamic?

24.3 Substance-Dependent Conversion Factors

***>>ToDo 71:* Re-check continuity, relevance.**

More generally, it might be useful to provide contextual defaults for other information in addition to the substance involved. In practice, the free interchange of mass and weight is justified by assuming that the measurement was done in a gravity similar to the Earth's. Other conversions are facilitated by assuming standard temperature, pressure, altitude, and other such characteristics. Yet another contextual default concerns default data and implied units; salaries are often expressed simply as money, with the implicit understanding that it is a Money/Time rate having an assumed time interval, such as a month.

24.4 Operating With Quasi-Dimensions

24.5 Input/Output

There is no such thing as "the quasi-dimension of a quantity" — all the properties of any individual quantity are associated with its dimension. Therefore, no special treatment needs to be given to the input or output of a quantity whose dimension is spanned by one or more quasi-dimensions.

25 Point Dimensions

***>>ToDo 72:* To be completed.**

25.1 Defining Point Dimensions

Define new point dimension and associate units. State related interval dimension.

25.2 Declaring Point Dimensions

25.3 Operating With Point Dimensions

25.4 Input/Output

26 Vector Dimensions

***>>ToDo 73:* To be completed.**

26.1 Defining Vector Dimensions

26.2 Declaring Vector-Dimensioned Data

Declare which vector dimension. Are coordinate systems associated with dimensions, or with variables? (I think the treatment of coordinate systems should parallel the treatment of units: associate with dimension, so that variable is of type "dimension," rather than type "dimension-in-coordinate-system").

26.3 Operating With Vector Dimensions

26.3.1 Compatibility

Two vectors are compatible for assignment, comparison, and binary arithmetic operations if and only if they have the same dimension. Inner and outer product have the usual requirements on rank. Inner product requires that elements to be added are of the same dimension. Operations between a scalar and a vector will be done element-by-element; the vector and scalar are compatible if each element of the vector is compatible with the scalar.

26.3.2 Coordinate Transformation

Operations between vectors require that the vectors be in the same coordinate system (this is parallel to the requirement for operations between SPQEs, that the units be the same). Most operations do not require any particular choice of coordinate system; those that do are specified below.

26.3.3 Comparison and Assignment

These operations are done element-by-element. The individual elements are treated just like any other SPQEs: compatibility is tested, unit conversion is performed as needed, etc. All this is assumed in the rest of this discussion.

26.3.4 Arithmetic

Vector addition and subtraction are done element-by-element.

***>>ToDo 74:* Questions about vector arithmetic.**

There does not seem to be any notion of element-by-element multiplication or division—is that right? (Multiplying/dividing by a scalar is as close as it comes - Stephanie).

Multiplication by a scalar is done element-by-element. Since the effect of this operation depends upon the coordinate system, a vector is first converted to its reference coordinate system (if necessary). We hypothesize that this will usually be the Cartesian coordinate system.

The dot product of two vectors is computed in the usual way, after first converting each vector to its reference coordinate system ***(is this right?)***. Since the dot product involves addition, all elements to be added must be of the same dimension, and that is the dimension of the result.

The cross product of two vectors is also computed in the usual way, after first converting each vector to its reference coordinate system. Its result, not being a vector, is outside the scope of this discussion.

26.4 Input/Output

27 Aggregate Dimensions

***>>ToDo 75:* To be completed.**

27.1 Defining Aggregate Dimensions

27.2 Declaring Aggregate Dimensions

27.3 Operating With Aggregate Dimensions

27.4 Input/Output

28 Accuracy, Precision and Context

***>>ToDo 76:* To be completed.**

Say what such support would consist of. Perhaps move some text from Section 34.5 and Section 34.6.

29 (Miscellany)

***>>ToDo 77:* Where should the stuff in this section go?**

29.1 Use of Systems

What it would be like to use such a system. What we'd say to it. What we'd see. Kinds of "user". What would ease the life of each kind of user.

29.2 Language Issues

Dependency on evaluation order. Binding of unit to quasi-dimension (convenience/flexibility tradeoff).

29.2.1 Recommended Functions and Features

<Haven't decided on this, but it would be a real feature of the paper (or a separate paper)>

29.3 Representation Issues

This list is imported from elsewhere. Weave it into the text.

1. Define plausible storage formats and techniques, with appropriate language for management.
2. What determines units in which data is stored? Other efficient storage formats?
3. Need to record original input form?
4. Note that there are infinitely many derived dimensions, so the type system should not assume it has a table of all dimensions.

29.3.1 Unit Conversion

29.3.1.1 Pairwise Conversion

Recall that a unit maps from a physical quantity to a number. Although some computing systems actually perform those mappings, they are outside the scope of this paper. We assume that physical quantities can be expressed somehow as PQEs, and are only concerned with converting them to other PQEs using other, given units.

The conversion from one unit to another can be represented in a variety of ways. Some candidates are:

1. $\psi = \alpha x$
2. $\psi = \alpha x + b$
3. $y = f(x)$, where f is a user-written function

The first two can be automatically inverted, while the third cannot. The first form is inadequate to deal with quantities like date and temperature. The second covers a wide variety of common units, but only the third can provide unusual mappings, such as abs(x), and others discussed elsewhere in this paper. These are:

- time-dependent conversions (used in converting currencies)
- asymmetric conversions (also used in converting currencies)
- 1/x (useful in converting fuel consumption to fuel efficiency)
- log(x) (useful in representing gain in db). Note that transcendental functions (generally, any mapping which is equivalent to a power series) can be applied only to dimensionless quantities, since otherwise the terms of the series are not compatible. However, one still might want to represent them, as there are many interesting dimensionless quantities.
- Months, days, and weeks corresponding to a 360-day year

We estimate that the second form is adequate for basic dimensional support.

Stephanie says: Be careful here - logarithmic units keep coming up as both important and difficult. We need to make sure that we understand the issues, and that we can handle logarithmic units using the y = f(x) form.

Bruce says: note that quantities expressed in logarithmic units violate the basic rule for inclusion in our discussion: measure(a#b) = measure(a) + measure(b). On the other hand, they are isomorphic to quantities which do obey it. I'm inclined to include them, but to note that quantities expressed this way need special treatment (either coercion into our domain, or special operators).

Schemes 1 and 2 above assume symmetric conversions, reflecting the notion that the amount of a quantity is not affected by the units chosen to express it. If a system stores conversions in a form which can be automatically inverted, this simplification can cut in half the effort to teach the system about a new unit, and the space to store conversion factors. The conversion mechanism could be generalized by not requiring the forward and reverse mappings to be inverses of one another. Someone who goes to the bank, for example, and converts currency from dollars to pounds and back, will wind up with less money than when he started. The conversion mechanism could also be generalized by allowing a mapping in one direction only (we can think of no use for this).

Conversion to or from a unit whose range is non-numeric is discussed in Section 34.11.

Again, non-numeric quantities are outside our domain of discussion. I'd like to keep Section 34.11, but note explicitly that it's outside our domain of discussion.

29.3.1.2 Conversion Within a Family

Consider the *conversion graph* for a dimension, in which a vertex corresponds to each unit in the dimension, and there is an edge from vertex u_i to u_j if we know how to convert an SQPE with units u_i to an SPQE with units u_j. In this discussion we assume the graph is undirected (but see Section 29.3.1.1).

A system can enforce a "star" configuration for the conversion graph:

- One unit per dimension is distinguished.
- Every time a unit is added, the conversion to and from the distinguished unit must be specified.
- No other conversions are specified.

This approach yields a connected graph with a minimum number of edges, but the resulting system sometimes uses two operations to convert from one unit to another, costing time and precision. If the system allows fewer conversions to be specified, the graph can become partitioned: it is then not always possible to convert from a unit to any other. We do not consider such systems further. If the system allows more conversions to be specified, then there is more than one path between some pair of units. In this case, a choice has to be made among the paths. The choice can be made to preserve the maximum precision of the conversion, to take the minimum time, or to optimize some other metric. Performance problems can occur if a particular conversion is common and the selected conversion path is long; providing the direct conversion is a solution. Another solution is to use a fully-connected graph, by calculating all direct conversion paths automatically.

The choice of unit used as the center of the star can have an externally-visible effect on precision and performance. The effect on precision is due to the fact that only a finite number of values can be represented exactly, and it is influenced by more than just the number of digits available. For an example which is easy to visualize, we'll use a very coarse resolution. Consider a system which represents only the odd integers, and which uses the meter as its internal unit of length. If we present a length of 40 feet to such a system and ask for it back, again represented in feet, what happens? The length is stored internally as 13 meters, and converted on output to 43 feet. Note that this is larger than the original length. If the same system used the yard as its internal unit of length (thus changing the precision only slightly), the same length would be reproduced as 39 feet, smaller than the original length.

A system which represented only the even integers would exhibit similar behavior, but the choice of internal unit would have the opposite effect: converting to meters and back would reduce the length, while converting to yards and back would increase it.

40 ft.	Yards	Meters
Odd integers	39 ft.	43 ft.
Even integers	42 ft.	39 ft.

Our example intentionally uses coarse resolution, but an example of this sort can be constructed for any choice of unit and numeric resolution. Whether this behavior is acceptable or not is application-dependent. Designers may wish to make the choice of unit used internally a configuration parameter.

29.3.2 Precision and Context

If we use a tuple form of implementation for PQEs, then we risk overloading and redefining the tuple equality and assignment operators if we take attributes like precision and context into account. (This is the same problem we ran into when comparing SPQEs for equality.)

29.3.3 Things to Represent

29.3.3.1 Physical quantities

Have we already said enough about PQEs, or is there more to say here?

29.3.3.2 Dimensions

Want to represent compound dimensions as product of powers of reference dimensions. That suggests assigning consecutive integers to reference dimensions. Also need to store their names, and names of their aliases. Probably want links to associated units.

29.3.3.3 Units

Probably represent:

- Associated dimension.
- If a star conversion structure, whether this unit is the center; if not, conversions to/from center.
- If not a star, conversion to/from each other unit for this dimension.
- Possibly the precision and accuracy of each conversion.
- Name, aliases.

29.3.3.4 Conversions

Adequately discussed above?

29.3.3.5 Coordinate systems

Haven't thought much about this. Is it enough to just provide the conversion matrix to/from Cartesian Coordinates for each one?

29.3.3.6 Numbers

By this we mean the numeric part of PQEs. We want to offer all the built-in numeric types, certainly. A system may wish to support richer numeric types than the underlying hardware supports (e.g., ratios, bignums), but we can treat those as if they are built-in. t makes sense to represent explicitly the precision of each numeric type (assume accuracy=precision).

Can either have a separate section on non-numeric PQs, or ***note that they are outside our domain of discussion.***

29.3.3.7 Precision, Accuracy

Currently done in various ways [2, 24, 30]. Choice may be fixed, administrator-configurable, or user-configurable. Arithmetic between two finite-precision quantities ideally needs a covariance matrix, which is associated with the pair, not with either one. Have not thought about how to specify that.

29.3.4 Levels of representation

29.3.4.1 Input

29.3.4.2 Output

29.3.4.3 Storage

Note that the numeric type chosen for storage of PQs may have a visible effect on the value and precision. This is a side-effect of the representation, and needs to be described so. There may be a semantics that associates a numeric part with a dimension or unit (e.g., count or percent), but we haven't discussed it.

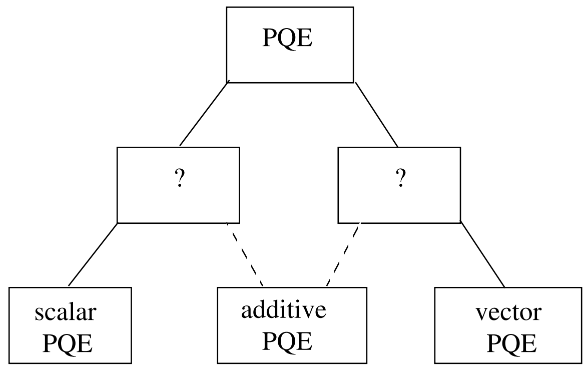
29.3.5 Additive Physical Quantity Expressions (APQEs)

Is is tempting to use the VPQE format to represent APQEs. For example, we might like to represent the APQE “5 feet, 5 inches” as the tuple <(5 feet), (5 inches)>. However, this representation is ambiguous, as it could represent either a scalar length of 65 inches or a point in 2-space. A system need not represent APQEs internally; but if it does, the representation must distinguish them from vectors.

The functions defined on APQEs are a mixture of those for vector PQEs, scalar PQEs, and neither:

- Dimension() returns a single dimension.
- Number_of_Elements() returns the number of elements.
- Unit() and Number() return a list (not a vector).

Since APQEs share properties of both scalar and vector PQEs, their taxonomy is ambiguous:



Since a system need not represent APQEs internally and we see no compelling reason to make the choice either way, we have left it unmade.

29.3.6 Other Representation Issues

The following text is imported from elsewhere. Weave it in here and above.

We said earlier that there are three attributes of an SPQE: dimension, units and number. It is relatively straightforward to determine representations for dimension and units (once the names of dimensions and their associated units are established), as we can simply use the string representation of the appropriate name. Alternatively, the unit component might be stored as an OID. There is also the option of omitting the units, and storing the PQE as only a number, although this might lose information if the units used were not the default units. In any case, most of the work to be done relative to representation surrounds the “number” attribute of a PQE.

Let’s consider how we would like to use a PQE. We would like to be able to view it, to store it, and additionally, we would like to be able to use a PQE for input when a physical quantity is required. For example, we might like to see the length of my car in scientific notation, and we might want to store the numeric component as a 64-bit floating point number. We might like to tell the system how long my car is by typing in something like “8.6667 feet”.

The example of the length of my car assumed some sort of floating point representation of the associated number, but there is no reason that integers can’t be used to represent magnitudes of physical quantities as well. Perhaps we might record temperatures in whole degrees only. We might even choose to view or store the length of my car in whole feet. (Clearly this would have an effect on the precision of the measurement we record, but it should be possible.)

The point here is that just as a physical quantity has no inherent units associated with it, a PQE has no associated numerical type (integer, float, double, etc.) associated with it. In fact, we can consider the specification of such a numeric type to be a mapping from a PQE to a particular representation, just as we consider a particular unit to be a mapping from a physical quantity to a PQE.

We have not yet addressed how these representation specifications are to be done. However, storage specifications will be handled separately from other issues, as we wish to separate interface from implementation as much as possible.

The remaining text in this section is taken from Bruce’s memo on I/O, slightly modified to fit here. We may want to fold it into the discussion above.

With respect to I/O, our recommendations here should minimally accomplish the following:

- Convince us and others that the I/O system allows input of all information required by the underlying machinery. For example, on input we need to know dimension, unit, and number. We need to determine whose responsibility it is to make sure that the underlying machinery knows the intended dimension. The input system must know:
  - what to do if the unit is missing or ambiguous
  - what to do if the number is missing
  - how much precision is intended in the number

  and we should make clear how these things are determined. We have a suggestion for how to handle missing units (see Section 22), and we can make other such suggestions for the number and precision information. Users must be able to ask for casts and operations on physical quantities; we probably have no opinion about how those should be requested, but it makes sense to list the casts and operations.
- Convince us and others that the I/O system allows output of all the information produced by the underlying machinery. For example, on output, we want to deliver a unit and a number, and possibly a dimension. There may be times when the I/O system will want to omit either, and this should be mentioned. Also, we may want to communicate the amount of precision in a number. Further, there is usually a choice of units to be output, and we may offer suggestions on the choice. Similarly, there is sometimes a choice of ways to specify a unit or dimension (e.g., "1/ohm" vs. "ampere per volt"). This subject is hard, like printing an algebraic expression in "simplest" form, but we may still have some advice.
- Where we have noticed non-obvious things useful to someone building a working system, this section should mention them.

  (Actually, I think that we should mention implementation issues wherever they come up, whether it's in I/O or elsewhere. Alternatively, we could collect all of these implementation observations in one section.)
- If we foresee functions common to the underlying machinery and the I/O system, we should mention those.
- We don't have to build the I/O system, nor even design it, but we should convince ourselves and others that it is possible to build a sensible implementation with the necessary capabilities.

29.3.7 Specialized and Quasi-Dimensions

We speculate that some aspects of the behavior of such specialized dimensions might be modeled using Ada's derived types (but see [19]), C++'s private inheritance, and the relational model's domains [12, 14]. See Section 34.10.

29.3.8 Expressions

One would like to be able to construct a literal (a PQE?) of any type supported by the system. For ordinary dimensions one simply constructs an ordinary PQE. For specialized dimensions, the language designer has a choice:

- Require the applicable units to be partitioned among the dimension and its specializations. Then the choice of unit in the PQE determines its type.
- Define the syntax of a PQE to allow an optional explicit dimension. If it's present, then the PQE is of that dimension. Note that its type is allowed to be that of any of the parent dimensions; by convention we say that it is exactly the given specialization.

There is an issue in determining the type of arithmetic expressions involving mixed ordinary and specialized and quasi-dimensions. The rules defining the specialized and quasi-dimensions will cover some of the cases, but they can't cover them all. ***[I think we decided the following:]*** In the remaining cases, the type of the expression is the same as would be formed if each specialized dimension or quasi-dimension were replaced by the root of its dimension tree. E.g., (Concentration in moles/liter) × (feet of Width) has type Molarity ×Length, and not Concentration ×Width.

29.4 Performance Issues

Compile-time checking vs. run-time checking. Unit conversion topologies.

Systems dealing with measurement data are likely to be handling very large volumes of data. Therefore, search speed over measured data must be very fast. This holds for not only the numeric and units parts of measured data, but it must also be possible to quickly search over associated context information.

30 Conclusion

>>ToDo 78: To be completed.

31 Acknowledgments

We would like to thank Rafiul Ahad, Jim Davis and Evan Kirshenbaum (of Hewlett-Packard) and Frank Olken (of Lawrence Berkeley Laboratory) for their suggestions and insights.

32 References

1. American Society for Testing and Materials (ASTM), "Standard Practice for Use of the International System of Units," E380-92, PCN 03-543-092-34, 1992.
2. American Society for Testing and Materials (ASTM), "Standard Practice for Using Significant Digits in Test Data to Determine Conformance with Specifications," E29-89, 1989.
3. ANSI/IEEE Std. 280-1985, "IEEE Standard Letter Symbols for Quantities Used in Electrical Science and Electrical Engineering.
4. ANSI/IEEE Std. 945-1984, "IEEE Recommended Practice for Preferred Metric Units for Use in Electrical and Electronics Science and Technology".
5. ANSI/IEEE Std. 260-1978, "IEEE Standard Letter Symbols for Units of Measurement".
6. Baldwin, Geoff, "Implementation of Physical Units", SIGPLAN Notices Vol. 22, No. 8, August 1987, pp.45-50.
7. Biedl, Albrecht, "An Extension of Programming Languages for Clerical Computation in Science and Engineering with Special Reference to Pascal", SIGPLAN Notices, April 1977.
8. Boring, Edwin G., "The Beginning and Growth of Measurement in Psychology," in *Quantification, A History of the Meaning of Measurement in the Natural and Social Sciences*, Harry Wolf, Ed., Bobbs-Merrill, 1961.
9. Bridgman, P.W. "Dimensional Analysis," Yale University Press, 1922.
10. Buckingham, E. "Phys. Rev. 4," 1914.
11. Cmelik, Robert F. and Narain H. Gehani, "Dimensional Analysis with C++", IEEE Software, 5/88, pp. 21-27.
12. Codd. E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* 13, No. 6, June 1970.

13. Coombs, Clyde H., “A Theory of Data, “Psychological Review, Vol 67, No. 3, pp. 143-159, May 1960.

14. Date, C. J., “What is a Domain?” in *Relational Database Writings 1985-1989*, Addison-Wesley 1990.

15. Deaves, J.C. and Pache, J.E., “Chemical and Numerical Indexing for the INSPEC Database”, The Indexer, Vol. 16, No. 3, 4/89, pp.163-167.

16. Dreiheller, A., Moerschbacher, M. and Mohr, B., “Programming Pascal with Physical Units”, SIGPLAN Notices, Vol. 21, No. 12, pp.114-123, December 1986.

17. Gehani, Narain H., “Units of Measure as a Data Attribute”, Computer Languages, Vol. 2, pp.93-111, 1977.

18. Gehani, Narain H., “Databases and Units of Measure”, IEEE Transactions on Software Engineering, Vol. SE-8, No. 6, November 1982.

19. Gehani, Narain H., “Ada’s Derived Types and Units of Measure,” Software—Practice and Experience, Vol. 15 No. 6 pp. 555-569, June 1985.

20. Hilfinger, Paul N., “An Ada Package for Dimensional Analysis”, ACM Transactions on Programming Languages and Systems, Vol. 10, No.2, 4/88, pp. 198-203.

21. Horvath, Ari L., “Conversion Tables of Units in Science and Engineering”, Elsevier Science Publishing Co., Inc., The MacMillan Press, Ltd., 1986, ISBN 0-444-01150-1.

22. House, R.T., “A Proposal for an Extended Form of Type Checking of Expressions”, The Computer Journal, Vol. 26, No. 4, 1983.

23. INSPEC, “Numerical Indexing: Description and Thesaurus”, IEEE, 1989 Edition.

24. ISO, “Guide to the Expression of Uncertainty in Measurement,” International Organization for Standardization, Geneva, Switzerland, 1993.

25. Isaacson, E. de St Q., and Isaacson, M. de St Q., “Dimensional Methods in Engineering and Physics“, John Wiley and Sons, New York 1975

26. Kahn, David L., “A Unit-conversion Algorithm”, BYTE 3/85, pp. 151-164.

27. Karr, Michael and Loveman, David B., “Incorporation of Units into Programming Languages”, Communications of the ACM, Vol. 21, No. 5, pp. 385-391, May 1978.

28. Kent, William, “My Height: A Model for Numeric Information”, working paper.

29. Maenner, R., “Strong Typing and Physical Units”, SIGPLAN Notices, Vol. 21, No. 2, pp.11-20, March 1986.

30. Taylor, Barry N. and Kuyatt, Chris, “Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results,” NIST Technical Note 1297, National Institute of Standards and Technology, Gaithersburg, MD, January 1993.

31. Wand, Mitchell and O’Keefe, Patrick, “Automatic Dimensional Inference”, in *Computational Logic: Essays in Honor of Alan Robinson*, Jean-Louis Lassez and Gordon Plotkin, Eds. pp. 479-483, MIT Press, 1991

VI OTHER POTENTIALLY USEFUL MATERIAL

[Some of this might be usable in the main text.]

33 Peculiar Measurement Domains

[This is a focused collection of observations about certain troublesome measurement domains. It may be partially redundant with scattered stuff in the main document (e.g., Section 2), but I think it could make interesting reading in its own right. I would try to find a place for this in the main document.]

33.1 Angles

[Maybe just a reference to Section 7.1, if it all stays there in one chunk.]

Rotational	Circular	Interior	Heading
360°>180°>0°	180°>0°=360°	360°=180°=0°	360°=0°≠ 180° x>y: error
280°+280°=560°	280°+280°=200°	280°+280°=160°	280°+280°: error

33.2 Temperature

[Collect (repeat?) a lot of the stuff scattered in the document. Maybe also refer to interesting discussions in other documents.]

33.3 Weight and Mass

***>>ToDo 79:** Describe their relationship.*

A plausible specialization graph (= means alias):

```
             |
Length/Time--|--Speed
             |

              |               |          |
Length/Time^2--|--Speed/Time--|--Accel--|--Gravity
              |               |          |

                   |              |
Mass*Length/Time^2--|--Mass*Accel--|--Mass*Gravity
                   |   =Force     |   =Weight
```

***>>ToDo 80:** A specialization graph question...*

*That graph also raises this question: if X is a specialization of Y, is X*Z automatically a specialization of Y*Z (and ditto for X/Z)? Is the question significant?*

33.4 Time and Date

Notes...

Pick up the treatment of dates in spreadsheets in Section 3.5.

Another observation: the internal representations of date in computers use many different origins. That representation confirms the intuition of a date as being a count of days since some origin point. "Day" is a well-behaved unit. (Tie in with the discussion of counts [Section 15.1].)

33.4.0.1 Introduction

This somewhat academic paper outlines the essential semantics of time, with which any temporal model should be consistent.

33.4.0.2 The Time Line

The best metaphor for time is a single straight line, analogous to the real number line but with no assumption as to where zero is.

Let's assume that we all live in the same "reality space" which has exactly one absolute objective time line. There may be other time lines in fiction, or in people's subjective perceptions, but we assume one time line.

An *event* is something that occurs at a point on the time line. Relativistic considerations are ignored. We assume that it makes sense to speak of synchronicity, i.e., two events occurring at the same time. Determining that to great precision may be quite difficult, but that's another matter. We also assume that time passes at the same rate for all observers.

The time line is *directed*, so there is a sense of *before* and *after* for any two distinct time points.

The time line is *measurable*, so that it is known whether one interval is shorter than another.

The time line is *continuous* and *unquantized*; there is no minimum time interval, and there are infinitiely many time points between any distinct time points.

Date and day-time are simply measurements of different granularity along the time line, much like miles and feet.

We take the basic unit of measurement to be the *day*, defined to be an interval on the time time line during which the planet Earth makes one complete revolution around its axis. Our model takes this to be a universal constant.

This interval is subdivided in the familiar ways into hours, minutes, seconds, and fractions of seconds, identified by numbers.

The boundaries of hours are synchronized on the time line (hence so are the boundaries of minutes and seconds), but the boundaries of days are not. There are twenty-four different zones in which the day begins at twenty-four different hour boundaries on the time line. (We ignore time zones which don't differ by an hour.) Hours are numbered from 0 to 23 within each day in each time zone. The concept of what day it is, or what time it is, is not defined on the time line, but only in each time zone. A given event on the time line might occur on different days and at different times in different time zones.

Time zones have geographic boundaries on Earth. The time zone in which the Greenwhich observatory is located is, by convention, used as a standard reference, denoted GMT (Greenwhich Mean Time).

Time zones are not fixed with respect to the time line. Twice a year they shift back and forth by one hour. On these occasions a day can have 23 or 25 hours.

Days are arbitrarily partitioned into groups of seven called a *week*, with each day in a week being given a different name. That's analogous to giving the twelve inches in a foot twelve names instead of numbers.

A "spot" is a fuzzy interval corresponding to the granularity (precision) of a time measurement. (A small spot might be called a "moment".) Dates, for example, identify spots whose duration is one day.

One of the main reasons for having a concept of time is to be able refer to specific spots on the time line. The only natural way is to refer to unique identifiable events, such as a memorable eruption of a certain volcano, or a memorable earthquake, or the birth or death of a particular person. This is not a very effective way to identify many time points to many people.

This is the reason for calendar systems, whose main purpose is to measure the distance on the time line between a given spot and some event chosen to serve as an arbitrary origin (zero) on the time line. Different calendar systems use different origins and different units of measurement, and they give different measurements in different time zones.

We customarily give precise measurements of large quantities in some convenient mixture of large and small units. Instead of expressing a certain distance in millions of inches, we express it in some convenient combination of miles, feet and inches. Would we tolerate a system in which different miles had different lengths, and so did different feet? Well, that's the way we measure large intervals.

A *date* is a measure of the distance between a day-long spot and some fixed origin spot. We measure it in years, months and days, analogous to miles, feet, and inches. Except that these units are not fixed, and we give some of them names instead of numbers.

The point in time identified as midnight of December 31, 1999 represents a very precise interval from the calendar origin, which can be counted as some definite number of days plus no hours, minutes, seconds, or fractions thereof. Since this point actually corresponds to twenty four different points on the time line, one for each time zone, it follows that each time zone has its own calendar origin, although they are all within 24 hours of each other on the time line.

Time of day measures an interval beginning at the start of the current day — within a given time zone.

33.4.0.3 Notes

How do we measure large intervals? If we do it in days or hours or subdivisions of these, the meaning is well defined. But exactly how long is ten years?

We often think of date and time as being essentially different things, but they are really just different units for measuring the same thing in large or small granularity, analogous to miles and inches.

(Of course, we've been saying "date" in the sense of an absolute date of July 4, 1776, rather than a cyclic date such as July 4.)

The fact that a date is represented by three data elements, i.e., three fields, has no great significance. It does not signify a relationship among three distinc entities, any more than a length whose measurement is recorded in miles, feet, and inches. The three data elements are an accidental byproduct of the choice of representation. It could just as easily be two elements (year and day within year) or one element (a large number of days).

Much of the difficulty with temporal data has to do with the wildly varying sizes of spots, and trying to make sense of ovelapping intervals bounded by such diverse spots.

33.4.1 About Time (August 1992)

33.4.1.1 Introduction

Time, especially dates, constitutes one of the most complex "attributes" we deal with in recorded data. Its exploration can shed some light on general problems of measured quantities, attributes, values, representations, and even entities and relationships.

Dates are sometimes perceived as attributes, having values in various representations. The representations themselves are complex, requiring elaborate conversion routines, and providing the primary motivation for such data constructs as compound fields. Dates are also sometimes perceived as ternary relationships, being a relationship among months, days, and years. This in turn exposes a number of ambiguities in our use of such terms. What is a month? In the sense of that ternary relationship, there are exactly twelve things in the domain of months (i.e., twelve things which can occur in the months' position of a date); thus there are twelve months, and "February" is the name of exactly one thing. On the other hand, a month is a time span of approximately thirty days (as in "I'll see you in a month"); and we might say that there exist an indefinite number of such things. Or we might say that a month is such a time span, but in such a way that they don't overlap; there are exactly twelve of them in any year. "February" is the name of one-twelfth of those things; in any century, there are 100 things named February. Or, finally, we might say that a month is exactly one particular such time interval: the month in which John F. Kennedy was elected is different from the month in which he died.

33.4.1.2 What Time Is

Our sense of "what time it is", or "what date", is an illusory absolute. It is in fact just as relativistic as our sense of place. Both senses give the illusion of absoluteness by virtue of certain arbitrary reference points. As we all know, there is no absolute coordinate system in the universe by which we can specify the location of objects. At best we can speak of some distances and other geometric relationships to other objects. We often forget this, although our sense of place is almost always relative to certain points on the Earth.

Quite similarly, the location of events in time must necessarily be relative. The date of an event is a measure of the time interval relative to some arbitrary reference event, that event in many cultures being the birth of Christ.

Our usual thoughtless notion of time refers to local time, rather than world time. We are likely to assign different birthdates to people born at the same moment on opposite sides of the international date line. In any model we have to decide how to deal with local time vs. world time, as with the assumption that events occurring on different dates necessarily occurred at different times.

The converse situation arises out of a question of resolution, or granularity. We tend to record the times of different kinds of events with different degrees of precision. Births and hirings are recorded to the nearest day; entertainment events and trains might be scheduled to the nearest minute; events of scientific interest might be recorded in millennia, or in nanoseconds. Something has to be worked out to accommodate these in a model with a single time-stamping mechanism. Shall we record all events to the maximum precision, or shall we allow the varying uncertainties we always allow in real life? Shall we continue to say that having the same birthday means being the same age, even though the births may have occurred almost 24 hours apart?

What should it mean to us to say that two events occurred at the same time? Local time, and variable granularities, make that question hard enough to deal with in our everyday world. The question becomes totally unmanageable for the likes of us if we consider relativity: modern physicists seem to be telling us that simultaneity is impossible to determine. In that context, we may begin to wonder what it means to record the time at which an event occurred. And if we really do want to account for relativity theory, then we had better accept that the interval between two events is subjective, being possibly different for different observers. The astronaut who returns to us ten years after launching may feel he's only been gone for a year.

Of course we all want to dismiss that as being too “far out” (pun?). But is it really going to be so long before we have to deal with such astronauts, or with on-board computers which return from space with different elapsed times from the ground-based systems? Will we have union arguments over whether that astronaut has earned one or ten years of salary and seniority for his one year of work? Will the astronaut object to being counted as nine years older, for retirement and insurance purposes?

It's not all so esoteric. The relevance keeps sneaking up on us. The relativistic question of synchronicity emerges in the problem of trying to synchronize events in distributed systems [5].

Time is inherently a difficult thing for us to think of by itself, or to describe directly. It is somehow a medium in which events occur. We perceive it as something which flows, with a sense of direction. It seems to fit closely the metaphor of a line (the time line), being an infinite and continuous one-dimensional thing, having a correspondence with the set of real numbers. The metaphor is augmented by a definite sense of direction: we have an indefinable intuitive sense of before and after, of past and future, which defines for us the universal convention for the direction in which time is increasing.

This is the best metaphor we can use for time in our work: a single directed infinite straight line. It may not be the ultimate metaphor. Relativity theory seems to posit multiple time lines. We will temporarily (note that temporal term!) ignore the difficulty of getting observers to agree as to where an event occurred on the time line (the synchronization problem of relativity theory, suddenly relevant to distributed systems). We should perhaps deal in subjective time lines, as perceived by people, hence a possibly different one for each person. But subjective time lines might not “measure” the same interval between two events. They might not even progress monotonically “forward”, if we take into account memories of the past, thoughts of the future, and various dreams and hallucinations [cite Borges]. Also, time travel may some day materialize out of science fiction, confounding our illusion of monotonic progression. And there are cultures (e.g., Hopi) which don't share our perception of time, differing in a way that we find virtually indescribable 6..

We must necessarily ignore such speculation, and content ourselves with the metaphor of a single directed line. The identification of points and the measurement of intervals along that line provide our best basis for dealing with times and dates.

33.4.1.3 Measurements on a Time Line

We use date and time units to express a variety of phenomena. Sometimes we refer to one particular point in time, such as the date a certain person was hired, or the time at which someone was born. Sometimes we refer to one particular time interval, such as the year 1979, or between 7 and 10 PM on June 1, 1979.

But sometimes it's not clear whether we're thinking of a point or an interval. We can only be sure an interval is intended when two points are indicated, as in March 15-18, 1979. When only one point is indicated, as in a date, that could really be intended as a point with coarse granularity, or it could be a reference to a 24-hour interval.

Sometimes we refer to an interval of a certain length, not fixed in any particular place on the time line — a year, 3 hours, etc.

And sometimes we refer to recurring points and intervals. January and Monday are such. So are partial dates, like the dates of holidays (Dec. 25th). So is time of day when no particular date is expressed or implied, and so are the times in train and plane schedules.

Thus we have points and intervals of time, sometimes fixed, sometimes floating, and sometimes recurring on the time line.

The intervals are measured in various granularities, or precisions. Date and time correspond to the precisions occurring most frequently in recorded data: time to the nearest day, and time to the nearest minute or second (or fraction thereof). Some other useful precisions include decades, centuries, and millennia.

Date and time are measurements of linear quantities, intervals on the time line, using irregular mixed radix systems. The measurements are quite analogous in most respects to the measurement of distance along a line. Hence, whatever techniques we know for data types, representations, and operations for distances should help us apply similar concepts to date and time.

33.4.1.3.1 Date

A date such as March 15, 1980 designates a point on the time line — a rather broad point, having a width of one day. The date identifies the point by giving its distance from an arbitrary origin, which is normally considered to be Dec. 31 of the year 1 BC. (There are a number of calendar systems in use in the world today. Unless otherwise indicated, dates will be described relative to the Gregorian calendar [3].)

A date expresses a "distance" along the time line in months, days, and years, just as an ordinary length might be expressed in yards, feet, and inches. That is, dates are simply linear measurements in a mixed radix (multiple units) system. March 15, 1980 could be re-phrased as a "distance" of 1980 years, 3 months, and 15 days from the origin. The months of the year are sub-units of a length, like the inches of a foot. The months happen to have names, while the inches do not.

There are some anomalies which render the analogy less than perfect. A coefficient in an ordinary linear measurement counts the number of whole units between the origin and the measured point, while for a date the coefficient also counts the unit which contains the point. A point occurring within the first foot of a line segment is counted as a distance of 0 feet, x inches, but a day occurring within the first month of a year is counted as month 1 (January), x days. Thus, to be more precise, the date March 15, 1980 actually represents a distance of 1979 years, 2 months, and 14 (and a fraction) days from the origin. There is no year 0; January 1 of the year 1 marks a point 0 years, 0 months, and 0 (and a fraction) days from the origin.

Another anomaly is the irregularity of the units. While the inches in a foot are all alike, the months of a year are not. There is not a constant multiplier for converting between months and days. So, the conversion between months and days is not a simple multiplication or division, but something like a table lookup or similarly complex algorithm. The analogy between inches in a foot and months in a year is more apparent if we use addition instead of multiplication: to convert x feet to inches, we sum the first x elements in the list:

    12, 12, 12, 12, 12, 12, 12, 12, ...

while to convert x months (of a year) into days we sum the first x elements in the list:

    31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31.

For leap years, 29 replaces 28 in the sequence. For arbitrary years, the conversion is ambiguous, although the sequence with 28 is conventionally the default.

Another anomaly has to do with negative values. With distances, all units of a measurement are counted in the same direction. For a point to the left of an origin, the yards, feet, and inches are all counted from right to left. Not so with dates. For negative, i.e. BC, dates, only the years are actually counted backwards. The months and days are counted forward in the same direction as AD dates. The date March 15, 10 BC corresponds to -10 years +3 months +15 days from the origin (or maybe that should be -9 years +2 months +14 days).

In spite of such anomalies, one basic concept should stand out: a date is simply a way of expressing a linear measurement in mixed units. It is not anything more mysterious or exotic. It is not, for example, a relationship among three entities (years, months and days), any more so than a length is a relationship among the "entities" yards, feet, and inches. Nor is it a distinct data type, unless we are also motivated to make distance measurement into another distinct data type.

33.4.1.3.2 Time of Day

Time of day is substantially the same phenomenon as date, being an interval along the time line, relative to some origin. The granularity is finer, and the origin concept is a bit different. In fact, there are two starting-point conventions: for a 24-hour clock, we measure time of day from the preceding midnight; for a 12-hour clock, we measure time of day from the preceding midnight or noon.

But, fortuitously, the anomalies associated with date are gone. Time of day is measured in a nice regular mixed radix number system, with no more or less anomalies than the measurements of distance. (Except that for the 12-hour clock we tend to use 12:xx instead of 0:xx in the first hour.)

In some ways, it is strange to think of date and time of day as being different phenomena. The specification of a fixed moment in time is often done in a series of six units: year, month, day, hour, minute, second (and fractions). The first three and the last three simply provide the high order and low order portions of the same measurement. Date in this sense is no more different from time of day than miles are different from feet and inches. It just happens that different conventions have arisen for their representation.

33.4.1.4 Computation Problems

It is most helpful to separate the concept of what a date means from the problems of its representations. The semantics of a date are simply an integer number of days from the origin day. Meaningful operations on dates are the same as the meaningful operations on distances from a fixed point on a line. A date can be increased or decreased by an integral number of days, and the difference between two dates is an integral number of days.

Similar operations can be defined for time of day and time intervals.

33.4.1.4.1 Time Zones and Daylight Saving Time

It is not the same time all over the world. An event on the time line maps into different date-time measurements in different places. In effect, each time zone measures time relative to a different origin point on the time line.

The measurement of time is therefore dependent on the place where it is measured. This becomes increasingly significant, of course, in the context of distributed systems and data bases.

Furthermore, clocks are set back or forward by one hour twice a year in some parts of the world, in observance of Daylight Saving Time (DST) (sometimes also called Summer Time). This also has the effect of shifting the origin point on the time line forward and backward by one hour, at various times of the year.

In a strange way, the measurement of time depends on the time at which it is measured. Some segments of the time line are measured relative to one origin, and some to another. To map an event into a date-time measurement, you not only have to know where on earth you are, but where you are on the time line. Time is a function of place and time:

$$T=f(P,T).$$

This dependence of time measurement on time and place affects the determination of:

- the time of an event,
- the date of an event,
- the interval between two events,

and causes a number of anomalies.

The date of an event can be affected by time zone and DST. For one party in a phone call it could be Monday while for the other party it is Tuesday. That could occur even within the same time zone around midnight, between regions which do and don't observe DST. In these cases, if both parties are maintaining a log of phone calls, they will be recorded as occurring on different dates.

Date and time are ambiguous references to the time line, unless supplemented with some knowledge of place. This knowledge of place determines both the time zone and whether or not DST might be in effect. Quite often we assume the “local place” as an implicit qualifier.

Given the date and time of two events, the computation of the interval between them can get complicated. Without taking time zones and DST into account, we might have a message arriving before it is sent. We might have events which actually occurred up to 25 hours apart being recorded with the same date. The elapsed time from midnight to midnight at a given location might be 23, 24, or 25 hours. (Does that constitute one day?) The interval between 2 and 3 AM on certain Sunday mornings might be zero hours, or two. (Are there two occurrences of 3 AM on some of those mornings? Can events occurring an hour apart occur at 3 AM?)

Birthdays and hire dates are slightly imprecise measures of relative age or seniority. People born at the same moment might have different birthdays, and people born up to 24 hours apart might have the same birthday. (That could even be 25 hours, with DST considered.) In fact, a person born at a later time could have an earlier birthdate. Such concerns might become non-trivial when inheritances, or layoffs, are at stake.

33.4.1.4.2 GMT

A common solution to such problems is to adopt a single global time reference for the time line, independent of time and place. That's precisely what Greenwich Mean Time (GMT) is. One difficulty with the solution is that it is not universally used, nor universally usable. We have a strong urge to keep our local senses of the appropriate times for sunrise and sunset and meals, and local time will continue to be used. Thus one needs to know whether an expressed time is local or GMT; and the algorithm for converting between local time and GMT still suffers from the anomalous time and place dependencies. Furthermore, conversion to GMT distorts dates. Recording birthdays in GMT would change the birthdays of possibly half the people in the world.

33.4.1.4.3 Irregular Units

Other minor computational problems have already been mentioned. Dates are not quite as well behaved as other mixed radix measurements because the sizes of the units are irregular. Months have different numbers of days, years have different numbers of days, and one notorious month has different numbers of days in different years.

Even the correct leap year computation is not widely known. Not every fourth year is a leap year; centuries are only leap years modulo 400 (three centuries out of four are not leap years).

33.4.1.5 Representation Problems

We have problems with the representation of date and time, though many of the problems are common to the representations of any measured quantities.

The basic phenomenon involved is the mapping of a measurable item to a character string. The mappings are complex, and many mappings are possible, yielding many possible character strings for the same item.

33.4.1.5.1 Mappings

The mapping of a measurable item to a character string involves choices in the following variables:

- Accuracy and precision
- Origin
- Units
- Coefficients
- Representation of units
- Representation of coefficients
- Sequence and punctuation

Accuracy and precision of measurement is something we will largely ignore here. We have enough difficulty with the representation of a single ideal measurement.

Choice of origin doesn't present a problem, when the boundaries of a measured item are defined. But occasionally they are not. In order to identify the location of some physical point, a reference point or coordinate system must be established. The measurement of distance to a city depends on the reference point chosen in the city. For date and time, the origin point is affected by the calendar system, the time zone, daylight saving time, and the choice between 12- and 24-hour clocks. Different calendar systems, such as the Jewish or Chinese, are based on origin points different from the Gregorian calendar in common use. Sometimes the origin is arbitrarily shifted: we lost ten days in the reform from the Julian to the Gregorian calendars, when Oct. 5, 1582 became Oct. 15, 1582.

For most measurements there are alternative sets of units available, such as metric and non-metric. For time (in the narrow sense), there seems to be only one set of units in general use: hours, minutes, and seconds. For dates (time in the broad sense), one set of units dominates: years, months, and days of the Gregorian calendar. But again, there are alternative calendars, with different lengths of months and years, sometimes based on lunar instead of solar cycles [3]. And we also have Julian notation, often used in data processing and quite unrelated to the Julian calendar, where the set of units is just years and days (analogous to measuring distances in just yards and inches).

Even for a given set of units and an idealized measurement, various sets of coefficients can be applied to the units. There is a “canonical form”

$$k_1U_1, k_2U_2, ..., k_nU_n$$

in which:

- for all terms except the first, k_iU_i is less than the next higher unit (i.e., don't write 13 inches),
- for all terms except the last, the coefficient is an integer,
- the coefficients k_i all have the same sign,
- the coefficient k_i corresponds to the number of whole units U_i between the measured point and the origin (possibly modulo some divisor).

Such canonical conventions are frequently violated. We often do write 15 inches instead of 1 foot, 3 inches; and 1.5 feet instead of 1 foot, 6 inches (or 4.5 feet instead of 1 yard, 1 foot, 6 inches).

As mentioned earlier, for dates BC, the years' coefficient has the opposite sign from the months' and days' coefficients. And for dates in general, the coefficients are all too large by 1.

For time of day on a 12-hour clock, the coefficient k_1 is always wrong: within the first hour after the origin, we write 12 instead of 0. And for dates we sometimes truncate the two or three high order digits of the years coefficient, writing 79 or 9 for 1979.

Representation of units is not usually of concern for stored data, since the units themselves are not stored but factored into the description: distance in miles, weight in tons, age in years. For time of day and date, even input and output forms don't seem to involve representation of units. But for time intervals, and other measured quantities, units can be represented in a variety of abbreviated or capitalized forms, and sometimes alternative symbols are used (such as ' for feet, or “ for minutes).

For the representation of coefficients, we have all the usual options concerning the representation of numeric quantities, such as number base, data type, precision, implied decimal points, leading zeroes, etc. For dates, the month coefficient might be an integer, or any assorted spellings, abbreviations, and capitalizations of the names of the months. A year might be given in Roman numerals, and clock faces sometimes have Roman numerals.

The problems of variability of sequence and punctuation seem to be peculiar to date and time of day. For dates, the year, month, and date coefficients can occur in just about any permutation. Occasionally a sign indication is included (AD or BC), sometimes at the beginning and sometimes at the end. Time of day always seems to be given in standard sequence, although it could have an origin indication (AM or PM) added. Time intervals, like most measurements, suffer the standard problem of omission of 0 coefficients: 2 hours and 10 seconds.

Punctuation conventions for time of day include various combinations of colons and periods (12:23:10, 12.23.10, 12:23.10 — the last having an ambiguous reference to either seconds or hundredths of a minute). For dates, punctuation conventions include various combinations of slashes, commas, periods, and blanks.

The net result of all these alternatives is an enormous variety of character strings corresponding to a single measured item.

33.4.1.5.2 Recognition Problems

The mapping of measured item to character string is not uniquely invertible. 3:10 might be hours and minutes or minutes and seconds. 3/10 might be March tenth or October third (or even 0.3). Some time and date conventions use periods, allowing 3.10 to have all four of those interpretations, in addition to being an ordinary decimal number.

Such problems are not unique to time and date. They are no different in principle from the problem of determining whether a sequence of digits is decimal or integer, and whether it might be in octal or hex. Similar solutions apply in all cases: impose arbitrary conventions to make them decidable.

Without such conventions, it is not possible to determine whether certain character strings are valid, nor how they compare, nor how to perform other computations on them.

<u>33.4.1.6 Modelling Time</u>

33.4.1.6.1 Ambiguities

The question of how time ought to be treated in information models must first deal with a sea of ambiguities. The terms involved have an unusually large variety of meanings and usages. A single term may correspond to a half dozen distinct concepts.

In this section we merely illustrate some of the ambiguities. We have made no attempt to disambiguate the terminology in the present paper; to do so would require the introduction of too many artificial terms and definitions to be intelligible. We hope that the context has made the meanings sufficiently clear.

Time:

1. A broad inclusive term, in the sense that date is a particular form of representing time. “A given date represents the time elapsed since the birth of Christ.”
2. Time of day, an aspect of time which is different from date, and is measured in a granularity of minutes, seconds, or fractions of seconds. “Date and time of birth.”

3. A time interval. “The flight time is 3 hours.”

Day:

1. One of: Sunday, Monday, ..., or Saturday. “On what day were you born?”
2. Relative day within a month. “Month, day, and year.”
3. Relative day within a year. “YYDDD is year and day in Julian notation.”
4. A single specific day. “On the day I was born, it rained.”
5. Any 24-hour period. “72 hours equals three days.”
6. A period from midnight to midnight. “Tom and Dick were born on the same day.” (Such a day could occasionally be 23 or 25 hours, due to DST.)
7. An interval between sunrise and sunset. “Night and day.”

Sunday:

1. One specific day.
2. The first day of any week. (Were two people born on Sunday born on the same day?)

Similar ambiguities apply to other intervals, e.g., week, month, year, January.

33.4.1.6.2 What “Is” a Date?

The question of how to model a date is really the same as the broader question of how to model any measured quantity. In terms of some common modelling constructs, we have a full range of choices: dates can be modelled as attributes, as entities, and even as relationships.

Dates as Attributes

For our purposes, we define an *attribute* as the name of some fact (such as “height” or “birthday”), and a *value* as a character string which can be associated with an attribute for some individual.

Values can be treated in two ways regarding equality. As pure character strings, unequal strings constitute unequal values. Thus “6 feet”, “6 ft.”, and “72 inches” are three unequal values of the height attribute; and “Jan. 1, 1979", “1 Jan 79", and “1/1/79” are three unequal values of the birthday attribute.

Alternatively, conversion algorithms can be introduced, allowing the inequalities of the previous examples to become equalities. Such conversion algorithms are not explicitly reflected as a formal part of currently known information models.

Dates as Entities

Whenever we allow for conversion among values of attributes, we implicitly acknowledge something invariant behind the different character strings so equated. There is something else there besides the characters in “6 feet”. There is a certain space, to which various representations of measurement can be associated. We can, if we wish, think of the thing being measured as an entity in itself. Thus a certain height, or a certain day, can be viewed as an entity.

The correspondence between such an entity and its various representations can be modelled in several ways. The simplest is to consider it an arbitrary mapping between entities and symbols, just like the mapping of people to their names or social security numbers, or the mapping of a certain color to the arbitrary synonyms “red” and “crimson”.

Alternatively, one could model the whole measurement phenomenon [4]. Various sets of units of measurement are introduced into the model, and the combination of a measured entity and a set of units is mapped into a set of numbers (the abstractions of the coefficients mentioned earlier). These numbers and the set of units are then further mapped into character strings taking into account the desired representations of the coefficients and units.

This last approach is generally too complex to be useful, although it is probably the most precise model of the phenomenon.

Treating measured quantities as entities is not a generally familiar concept — except, perhaps, in the domain of time. Certain cycles and patterns in our lives have become so culturally ingrained that they seem almost palpably to be entities. Days, weeks, months, and years seem to have that character. It does not seem so unnatural to think of a certain day as an entity, having various attributes and being related to various other entities. It had various rainfalls and temperatures in various places. There are people who were born, hired, fired, married, and divorced on that day. Wars were declared, battles fought, and treaties signed on that day. And so on.

Of course, there is a curious lack of objectivity about such entities. Different calendar systems carve out different entities (years, months) from the time line. A day is nothing more than the period of one revolution of a certain spinning object; the individual revolutions of other objects do not seem particularly to be entities. It doesn't matter. Entities are where we perceive them. They exist wherever we agree to think so [4].

We can contemplate the "naming" of intervals.

A year is an interval subdivided into 12 shorter intervals, the months. A week is an interval subdivided into 7 shorter intervals, the days. Why have we given distinct names to those shorter intervals?

A foot is an interval subdivided into 12 shorter intervals, the inches. Why haven't we given distinct names to those shorter intervals? Why don't we have a distinct name and concept of the 5th inch of every foot, just as we name the 5th day of the week or the 5th month of the year? Anybody who thinks Thursday is a distinct entity should also agree that the 5th inch of all feet constitutes one distinct entity.

Dates as Relationships

Because a date like March 15, 1979 is often recorded in three columns of data, it is sometimes held to be a relationship among three entities. It can be awkward to define what those entities are.

"March" here names an abstraction of the third month of all years, which is or isn't an entity just as much as the third inch of all feet is an entity. "15" names the 15th day of all months; it is the only "entity" which is common to the two "relationships" March 15, 1979 and April 15, 1799.

33.4.1.6.3 Time in Data Bases

Dealing with time in a data base involves all the concerns about representation and modelling already discussed. In addition, we must consider how time-related information about the real world is captured in data bases. (Other references addressing this point are included at the end of this paper.)

Real World States and Data Base States

Though often claimed, it is rarely true that a database is a snapshot of a state of the real world. Many databases include a great deal of historical information.

There are two sequences of states: the real world and the database. A single state of the database describes many states of the real world. That's precisely the significance of memory: information about many past states of the real world can be retrieved from a single present state of the data base. A database update creates a new database state, which may not correspond simply with a new (later) real world state. The new database state might reflect

- A deletion of an old real world state (purge expired information).
- An extension to include a new real world state (report of change occurring in the real world).

- A modification to some prior state of the real world (either an error correction, or a late report of an old change).

Earlier states of a data base are never altered. Earlier states of the recorded real world are often altered, in later data base states.

"As of" data retrievals are ambiguous: "as of" in the real world, or "as of" in the data base? Do you want your bank balance as of March 15, 1979, or as it appeared in our records on that date? While the former may be the more sensible interpretation, it could give different answers when asked at different times.

A data base state can not usually be described as a simple union of real world states. That would imply that the histories of different attributes and relationships were all maintained with the same granularity, reporting cycle, and persistence.

Sequentiality

There are at least two sets of time values: when things occur, and when they are recorded. Things are usually recorded later than their occurrence. And they are not always recorded in the same sequence as they occurred.

One of the open questions in information modelling is whether time should be introduced as a special construct, or whether it can be handled simply as another attribute defined and managed by users as needed. Because of the computational complexities mentioned earlier, it is a convenience to provide special facilities for time and date. On the other hand, there is little the system can do to automatically maintain time related information, unless users are willing to equate time of occurrence with time of recording. If time of occurrence has to be provided explicitly as part of the input information, then it seems to have the same character as other attributes included in the information.

Note that if dependences are automatically maintained in the data base (especially implications), then we must be concerned about recording events in the chronological order of occurrence. Otherwise there could be an enormous maintenance problem. If an event is recorded as having occurred earlier than some other events already recorded, then it may be necessary to "roll back" time, and recompute history. In the worst cases, previously generated information (or other output actions) may become invalidated, and perhaps previously accepted inputs may be rendered unacceptable (inconsistent with the newly perceived state of affairs at that time).

The problem goes even beyond a need to enter changes into the system in correct order. In a distributed system, if the changes are entered at different nodes of the network, they might arrive at the node holding the data in a different order than they entered the network.

33.4.1.7 Idiosyncrasies

Time also seems to be rich in phenomena illustrating how illusory and artificial are some of our perceptions of reality. There are so many things we perceive as "natural" and "existing" which, if examined very objectively, turn out to be surprisingly elaborate artificial conventions of our making.

The concept of time collects around it an amazing variety of mental exercises illustrating all sorts of perversities of which the human mind is capable. It is the vehicle by which many subtle mental abilities are learnt, and for which we invent many kinds of abstractions.

Time provides a general lesson in irregularities. Learning about time does wonderful things to the naive mind. They suddenly come into focus when you try to teach these things to a child. It seems to be a basic mechanism for bringing the mind in touch with many peculiar devices we unthinkingly use in dealing with the real world.

Reinterpretation of signs: "9" signifies 45, or 15, or a quarter.

The third hand is the second hand.

Abstraction of concepts: an angle between radial pointers is "the same as" a digital display.

Further abstraction: some clocks don't have numbers on them.

Metaphor: hands pointing.

First lesson in unnatural arithmetic: 10 + 4 = 2.

Ten can be before (earlier than) two.

International date line: is it the same day everywhere? The same time?

If you and I were born on Monday, were we born on the same day?

If you and I were born at the same instant on opposite sides of the international date line, were we born on the same day?

33.4.1.8 Conclusions

Date and time are complex and esoteric topics to contemplate, but their modelling is less tortuous. They can be treated largely as special cases of measured quantities, with some computational aberrations to be taken into account. Representation problems require arbitrary solutions, such as limiting the variety of forms allowed and introducing appropriate syntax to remove ambiguities. While arguments may arise over various such solutions, they are not conceptually difficult.

33.4.1.9 References

1. B. Breutmann, E. Falkenberg, and R. Mauer, "CSL: A Language for Defining Conceptual Schemas", in G. Bracchi and G.M. Nijssen, *Data Base Architecture*, North Holland, 1979.
2. J.A. Bubenko, "The Temporal Dimension in Information Modelling", in G.M. Nijssen, *Architecture and Models in Data Base Management Systems*, North Holland, 1977.
3. "Calendars", *Encyclopedia Brittanica*.
4. W. Kent, *Data and Reality*, North Holland, 1978.
5. L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Comm. ACM 7 (21), July 1978.
6. Benjamin Lee Whorf, *Language, Thought, and Reality*, MIT, 1956.

34 Miscellany

34.1 Types of Physical Quantities

It is necessary to be able to identify, refer to, and differentiate various types of physical quantities (domains) such as weight, mass, length, area, volume, time points and intervals, speed, velocity, acceleration, temperature, frequency, etc.

Identifying and differentiating types of physical quantities is a non-trivial task. Some, like length and weight, are reasonably simple concepts to identify and differentiate. Even these, though, can be troublesome if we want to be very careful, such as the distinction between weight and mass, or the distinction between weight in vacuum and weight in air.

Others, like locations in space or points in time, might raise a question as to whether they should even be considered physical quantities, since they don't seem to have a sense of magnitude inherently associated with them. A place is just a place, not anything which has a bigness to be measured.

The well-behaved ones are relatively precise concepts, often expressible in algebraic relationships such as area being the square of length, or speed being the ratio of distance to time. Others, like amount, while being a reasonably clear concept, are more like a family of closely related physical concepts. An amount of stuff might be expressed in terms of weight (iron), volume (water), area (carpet), length (rope), or even a simple numeric count (people, atoms). The amount of a given chunk of stuff is often expressible in several such terms. Is "amount" a single domain or several?

Other such problematic domains also involve the notion of amount. Concentration, for example, is essentially the ratio of two amounts, each expressible in any of the forms that amounts can take. Productivity is a ratio of amount to time, fuel consumption might be a ratio of amount to distance or time, and so on.

We need to be clear as to whether we consider amount to be a single type of physical quantity or a family of such types. If it's a family, does the single family behave in some ways like a single type? E.g., units which measure the same type of physical quantity can be converted to one another. Can the same be said of units which measure the same family of physical quantities? Can a weight be compared with a volume?

Amounts illustrate concepts which are not algebraically equivalent (e.g., weight and volume) yet seem to be the same in some sense. Conversely, some things which are algebraically equivalent don't seem to be the same. Are length and width the same type of physical quantity? What about the length of an object vs. the distance something travels? Is the production rate of rope or licorice in feet per hour the same concept as speed? Dimensional analysis treats angles as being algebraically equivalent to a ratio of two lengths, yielding a dimensionless quantity if we do cancellation. Does that make angle the same as a pure number? Then what do degrees and radians measure? Torque and work are both algebraically equivalent to Force times distance, but these are clearly not the same physical quantity.

Populations which do or don't include negative quantities constitute different types. Weight in a vacuum has no negative quantities, while weight in air does. Weight in air is really a measure of buoyancy. A helium balloon would pull upward on a scale, and lighten the weight of a person holding one.

It may also be useful to clarify whether a domain need be limited to quantities which can exist in reality. Dimensional systems are used not only for measuring real phenomena, but also for describing hypothetical or fantastic situations. Does the speed domain include speeds greater than the speed of light? Should 2c be an invalid speed expression? Does the energy domain include half a quantum? Then perhaps it might also be meaningful in some contexts to contemplate negative masses, and negative temperatures. ***(Illustrates the point that users of a measurement system need to establish consensus on the nature of the domains involved.)***

Language doesn't always help. Sometimes natural language fuses different concepts into a single term (as with the several domains called "angle"), and sometimes makes artificial distinctions within the same concept. Compare the notions of time and distance. They are quite similar, both being isomorphic to the one-dimensional real-number line. They become even more nearly identical if we imagine ourselves to be moving at constant speed in a straight line. There is a sense of place in both domains; "here" and "now" are isomorphic concepts. Both domains also have a notion of interval.

Yet there are strange asymmetries in the language we use in the two areas. With distance, we use distinct terms for the notions of point and interval, one being "location" and the other being "distance". Where I am is a location; how far I am from you is a distance. However, we use the term "time" for both point and interval: when I was born, and how long I've lived, are both considered to be notions of time. If we go by linguistic conventions, then point and interval seem to be different physical quantities in relation to distance but the same physical quantity in relation to time. (We do have the word "duration" for a time interval, but we don't have a corresponding word that makes a single concept of the phrase "point in time". ***(Jim suggested "instant", but it doesn't have exactly the right connotation. It may have something to do with an implied granularity. "In my chair" and "in California" are both location concepts, despite the disparity in granularity. In contrast, it seems comfortable to call "3 PM today" an instant, but not "today", even though both may be a description of when something occurs. To be resolved.)***)

On the other hand, Distance has one set of terminology for all granularities, while Time uses two. Whether we say that I'm in my chair or in California, both are considered to be "location". However, if we say that I was born at 3 PM or on February 19, one is considered to be "Time" while the other is "date".

There is another difference. For times we have distinct names for periodically repeating points and intervals, such as Tuesday, 3 PM, January, and January 15. We don't seem to have a natural counterpart for distance. It may just be an accident of astrophysics, in that our notions of time derive from the behavior of the spinning and rotating body we live on. If planet Earth moved in a straight line through space, without rotating around the sun or spinning on its axis, our notions of time and distance might be more isomorphic, with no natural notion of cycles in either.

The bottom line is that we can't always rely in our intuitions, common sense, or linguistics to guide an objective identification and differentiation of the types of physical quantities. We need some other basis for a systematic formal model of physical quantities and measurement.

34.2 The Reference Rocks

Symbolic representations of phenomena in a scalar measurement domain D are expressed in terms of units. A *unit* is defined as some instance u of D such that any instance of D is equal to a combination of some number of occurrences of u.

For example, we can pick a certain rock as the Reference Rock. We can find a lot of rocks each of whose weight is equal to the weight of the Reference Rock. For an arbitrary rock, we might find that its weight is equal to the weight of k rocks, each of whose weights is equal to the weight of the Reference Rock. We then say that this rock weighs the same as k Reference Rocks, or that its weight is k Reference Rocks.

If the weight of a rock isn't an integral multiple of the weight of the Reference Rock, we can generalize to rational numbers by saying that the weight of a rock equals the weight of n/m Reference Rocks if the weight of m of the rocks equals the weight of n Reference Rocks. If the weight of a rock isn't a rational multiple of the weight of the Reference Rock, it can be approximated by the weight of another rock that weighs nearly the same as the rock we want to weigh.

Thus the weight of any rock is equal to or approximated by the weight of k Reference Rocks, where k is some rational number.

That's what measurement is. It is rooted in counting: how many reference units are equal to the physical quantity being measured? Notice the close association between two meanings of the word "unit". The unit of measure is the size of the things we are counting; it corresponds to a count of one, i.e., a unit.

Different units are trivially illustrated by introducing reference rocks whose weights are not equal to each other. Let's say, though, that all Blue Reference Rocks weigh the same, and that all Green Reference Rocks weigh the same. Then a given rock might weigh the same as x BRR's or y GRR's. If we know how many Blue Reference Rocks weigh the same as one Green Reference Rock (or vice versa), then we know how to convert between x and y. Given either, we can compute the other.

34.3 Combinational Closure

This breaks down for temperature and location. I don't get any good intuition as to what it means to combine the temperatures of two kettles of water and judging that to be equal to the temperature of a third kettle. Nor do I get a good intuition of what it means to combine the locations of two points in space and judging that to be equal to the location of some other point, in the absence of a superimposed coordinate system.

Formal description of the combinational operator for a measurable quantity can be complex. Combining lengths, for example, involves a notion of laying end to end. Things which are not straight (perimeters of circles, coastlines of countries) must first be mapped into something straight before being combined.

Colors have such a combinational operator, but spatial locations don't.

Weights (masses) combine by inert juxtaposition. Just because anti-matter might annihilate matter when they come in physical contact doesn't necessarily mean that anti-matter has negative weight (or mass). This is analogous to saying that the sum of masses of some atomic particles is not given by the mass of the results of their fusion, since some mass might vanish into energy.

(However, we do use the opposite reasoning for particle charges, where we do think of them as being positive and negative because they cancel each other out. The real message is that this is at least a point of potential ambiguity which needs clarification in any particular measurement system before we can agree on what sorts of operations are valid, and what their results are.)

34.4 Division and Rational Closure

The preceding axioms don't guarantee closure under division. What we have established is that, given a reference weight x_0, *if* there is a certain weight x_1, then it can be expressed as (or approximated by, if continuity holds) kx_0 for some k. However, nothing in the axioms seems to imply that there must exist a weight equal to kx_0 for arbitrary x_0 and k. Closure is guaranteed for an existing x_0 and an arbitrary *integer* k, due to the closure property of # [Section 11.1.4], but not for arbitrary rational k.

Thus closure under division seems to be an independent axiom. This also doesn't seem especially important, until we find some domain where such closure doesn't hold. We simply make the observation in passing; in most cases it will simply be postulated that this axiom holds.

34.5 Information About Physical Quantities: *Measurements*

In the previous section we talked only about physical quantities, and not about how to describe them to computing systems (nor how computing systems might describe them to their clients). In this section we discuss a realm slightly different from the realm of physical quantities: the results of mapping a physical quantity into information about the quantity. Such results are found in the output of measuring instruments, in repositories of measurement data, and in the output of computations on such data. Most computing systems operate in this realm, although they largely ignore it.

The word "measurement" is commonly used to mean both the process of measuring something and the results of measuring something. Since we are largely uninterested in the process, in this paper we will use the word "measurement" to mean the results of measuring something. We also use it, more loosely, to mean the description of physical quantities, whether or not they have actually been measured. Thus the thickness of a wall to be built, the average of several temperature readings, and the result we expect when we measure the mass of a truck, are all loosely called "measurements."

34.6 Information About Measurements: *Measurement Data*

In Section 34.5, we noted that observations of the physical world usually have context, and finite accuracy and precision. In general, when those measurements are communicated or stored, further context is added and further precision and accuracy are lost.

These changes propagate into the effects which precision, accuracy and context have on arithmetic, assignment and equality. The effects are the same as described in Sections 3.6 and 3.7, above. Let us be careful, though, to separate the precision, accuracy and context of an observation's representation from those of the observation itself.

- An observation made with high precision may be stored with low precision.

- Bias can be introduced (by a rounding algorithm, for example) into an accurate observation.
- The act of presenting an observation to a computing system is an event with its own context (date, operator, location, etc.), separate from the context surrounding the observation. Two separate data, with different precision, accuracy or context, may describe the same observation.

34.7 Measurement Fundamentals

The preceding sections introduced the relationship between dimensions and units. This section gives a more rigorous basis to both notions, and further characterizes the kinds of quantity that would benefit from the support we describe in this paper.

Even at a rigorous level, the abstract phenomenon described by a dimension is still a matter of convention. If two people disagree about what mass is, there is no physical reference to which they can appeal. Even if they agree that properties such as inertia and gravitational attraction are the defining ones (and not, say, temperature or size), people may still disagree about whether the mass of a thing includes oxidation on its surface, etc., and whether physically impossible values (negative, or very large) are still members of the dimension. In this paper we assume agreement on such matters.

Given such agreement, there are different abstract paradigms for assigning a symbolic representation to a member of a dimension—by judging, polling, testing, measuring, etc. We will have more to say about these; for now we assume agreement on a paradigm.

Given an abstract paradigm, there are many possible implementations of it, with associated issues of isolating variables, repeatability of measurement, averaging of readings, etc. ***Give better examples of implementation issues.*** We do not address these physical properties, but focus on the abstract meaning of measuring an agreed-upon phenomenon. The subject of this paper—dimensioned data—are then the results of such measurements.

The sense we mean of "measurement" chooses a particular instance of some dimension as a "unit." Everything in the dimension is then expressible as having magnitude equal to k of the unit thing. The number k will be the symbolic representation for magnitudes in the dimension; obviously, its value depends on the particular choice of unit.

This sense of measurement and unit requires a notion of equality of magnitudes in the dimension. As with agreement on the meaning of membership in a dimension, we assume a notion of equality of magnitudes. Note that this is distinct from the notion of *sameness*. If Dan and Bruce are equally tall, it doesn't matter whether between them they have two heights or one; what matters is whether their heights have equal magnitude. In the rest of this paper we will use "equality" to mean equality of magnitudes.

We have two goals for the sorts of measurement we have described:

- Symbolic Notation: a mapping from an abstract phenomenon (such as viscosity or length) to some symbolic representation. The representation may be numeric, but need not be: school letter grades and category names such as "low," "medium" and "high" can be measurements.
- Operations. The reason for all this trouble is that representations are easier to operate on than the things themselves. Operations (such as addition and test for equality) on the representations should thus mimic the related operations (such as combination and test for equality) on physical quantities. It is again subjective which aspects of the behavior are relevant and what level of mimicry will do. We have more to say about this below.

Provided, then, with subjective notions of dimension, equality and relevant behavior, and focussing on the meaning and results of measurements rather than the physical process of obtaining them, let us describe carefully what properties a quantity must have to be measured, and what we can say about its measurements.

34.8 General Functionality

34.8.1 Physical Quantities

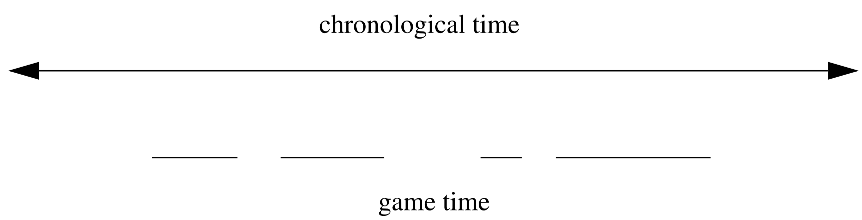
34.8.1.1 Count

Count is, in a sense, a degenerate unit, mapping a quantity (a set of things) into a number. To put it another way, units are a mechanism for converting amorphous things into things which can be counted, e.g., “how many” inches or pounds. Count can also be regarded as the numerator in some of the compound units expressed as 1/seconds, e.g., frequency (cycles/second) and radiation (disintegrations/second).

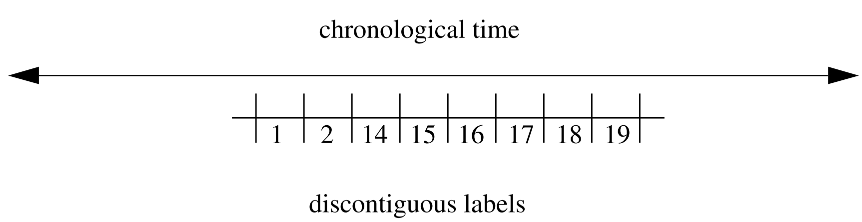
Count raises issues bearing some resemblance to the problems of substance-dependent conversions. It goes beyond compatible dimensions into concerns about “compatible properties”; we might not just ask whether two counts can be added, but whether it makes sense to add the numbers of apples to the number of oranges. But it we stray too far in that direction, we will ask whether it is valid to add the length of a shoe to the length of a car. In other ways, count is well-behaved: we can measure person-months of effort, or passenger-miles of travel. Interesting problems remain to be investigated along these lines.

34.8.1.2 Discontiguous Mappings

In some cases, the physical quantity measured is discontiguous. For example, the time during which a football game is actually played is discontiguous when compared to chronological time. Nonetheless, a touchdown will be described as happening “5 minutes into the second quarter” rather than at 3:13 pm. This corresponds to the following situation:



The reverse situation occurs when unit labels are discontinuous while the physical quantity remains continuous. For example, when switching from the Julian calendar to the Gregorian calendar in September, 1752, several days were simply omitted, thus producing a discontinuous labeling of calendar days:



34.9 Future Work

34.9.1 Semantics

Specialized dimensions, quasi-dimensions, and “dimensionless” quantities need exploring. We need a clean statement of their behavior and a sound place for them in the type hierarchy. We need to figure out how to extend the notions of dimensional analysis to specialized and quasi-dimensions. We would like also to characterize more fully the quantities which need to be represented as point + interval. Can we decide algorithmically which they are? For each one, is there a coordinate system with a natural zero?

Once that is in place, we want to look at vector dimensions of all those things, be sure they behave consistently, and verify our hypothesis that Cartesian coordinates is the proper canonical coordinate system for them.

It should be possible to sketch a small but nontrivial treatment of precision, accuracy, and context information. Precision can be integrated with the numeric format of PQEs. It may be possible to unify context with substance-specific conversions, by treating the type of substance involved and its relevant properties as "context."

34.9.2 Machinery

The support needed to add, change and delete dimensions, units, conversions, specialized dimensions, quasi-dimensions, and vector dimensions should be designed in more detail. Similarly, input and output of PQEs for non-basic dimensions (including the treatment of default units) needs more work.

We want to work out the compatibility issues for unary and n-ary operations, as we have done for binary operations, and to see what is required to support static type-checking.

Finally, we want to find a prototyping vehicle that can support the ideas presented here.

34.10 Domains

Some mechanisms currently exist for supporting domains. C++ provides this functionality through the use of private inheritance. For example, we could define the class Length as follows:

```
class Length {
private:
    // implementation
public:
    Length& operator+(Length&);
    Length& operator-(Length&);
};
```

Then we could define the subclass Height as follows:

```
class Height: private Length {
public:
    Height& operator+(Height& h) {
        return (Height)this->Length::operator+(h);
    }
    Height& operator-(Height&) {
        return (Height)this->Length::operator-(h);
    }
};
```

Private inheritance means that from the user's perspective, Height is not a subclass of Length. If we had not explicitly defined the + an d- operators in class Height, adding or subtracting two heights would not be legal. That is, private inheritance prevents a subclass instance from being treated as a superclass instance, and thus using methods defined on the superclass. Notice that the implementation of the class Height *can* access its superclass Length; however, this is hidden from the user.

In the case that a user really did want to add a Height and a Length (or perhaps a Height and a Width), it is possible to provide explicit conversion functions that would convert a Height or a Length into a Width. This would allow statements such as:

```
main() {
    Length l1, l2;Height h1, h2;
    Width w1, w2;
    w1 = convert(l1) + w2; // invokes Width operator +
    w2 = convert(l1) + convert(h1); // invokes Width operator +
    h2 = h1 + h1; // invokes Height operator +
w1 = l1 + w2; // illegal
w1 = l1 + h1; // illegal
```

Include also SQL3 and Ada examples?

34.11 Conversion of Non-Numeric Units

The range of a unit can be any totally-ordered set. Let U and V be such sets. The difficulties in converting PQEs between U and V are:

- There may be values represented in U that are have no corresponding representation in V (the mapping from V to U may not be *onto*). One example of this is in sporting events which keep track of events by the game clock. There is no "game time" for anything that happens while the clock is stopped. Another example is screw sizes: a screw that is physically between standard sizes has no "size."
- Suppose U has more elements than V, even though they cover the same space. Then a conversion from U to V loses information. For example, if letter grades A-C map to Pass and grades D-F map to Fail, then information is lost in a conversion from B to Pass: the conversion from Pass back to A-C is ambiguous.
- Even though U and V cover the same space and have the same number of elements, if the boundaries between categories in U and V do not line up, then the conversion between U and V loses information and the conversion back is ambiguous. ***Draw a picture.***

Note that some of these difficulties are present even when one set is the rational numbers. Thus, a conversion mechanism will encounter difficulties whether it employs a "star" conversion topology, with the rationals at the center, or a point-to-point topology.

These ambiguities will result in a loss of precision. There is a spectrum of loss, from the spacing between machine-representable numbers to 100%, and an equally wide spectrum of tolerance among users for such loss. There is thus no single solution to these difficulties.

The incomplete and ambiguous mappings cannot be performed automatically. A system must either refuse to perform these, or ask users to provide their own conversion routines.

34.12 Absolute Significance of Relative Magnitude

This is meant to support Section 34.7, but I'm not sure of its place.

Bridgman [9] states that any quantification scheme subject to dimensional analysis must have the property that if a unit twice as large is chosen, the resulting numbers are half as large. St Q Isaacson [25] restate this as: the relative magnitude of two quantities must be independent of the unit used. Both authors call this the principle of "absolute significance of relative magnitude." If a scheme for mapping quantities to symbolic representations obeys this principle, then any dimension can be represented as a product of powers of the reference dimensions.

From Bridgman we have:

1. measure(x, nu) = 1/n measure(x, u)

for all x and u. Let $u_1 = n*u_2$. Then:

2. measure(x, u_1) / measure(y, u_1) = (1/n * measure(x, u_2))/(1/n * measure(y, u_2))

   = measure(x, u_2) / measure(y, u_2)

This is St Q Isaacson's principle.

We can also prove a sort of linearity property. From the definition of measurement via units we have:

3. measure(x, u) = 1/measure(u, x)

thus

4. measure(y*x, u) = 1/measure(u, y*x)

   = 1/(1/y * measure(u, x))

   = y / measure(u, x)

   = y * measure(x, u)

[I suspect this follows if we know that # is associative and commutative. See Section 11.1.9. Bill.]

34.13 SI Conformance

One can build a system with selectable levels of conformance to the guidelines in [1]. At minimum it would recognize all the SI dimensions and units. At another level it might generate outputs which always follow SI recommended practice. At a still stricter level, it might reject input which was not in conformance with SI recommended practice.

34.14 Administration of Systems

This section needs more structure. Organize according to operations or operators? (hopefully not both). The more I look at this, the less I think it adds to the paper. Maybe this section should simply acknowledge the existence of this level of configurability, say that a system will need to allow an administrator to read/write/add/delete/set-default for each thing at that level of configurability, and stop there.

There are at least three levels at which a system supporting dimensioned data might be configurable. First, some things will be built-in (not configurable). Examples of these are:

- The compatibility rules for ordinary dimensions.
- A basis set of dimensions and units.
- Whether a system supports specialized dimensions.

Second, other things may be configurable, but only by an administrator, not by users (operators or programmers). Examples of these are:

- Dimensions and units beyond the basis set.

- Whether a system supports specialized dimensions.
- The compatibility rules for specialized dimensions.
- Default dimensions.

Third, still other things may be configurable by users. Examples of these are:

- Default dimensions.
- Dimensions and units beyond the basis set.

In this section we discuss the things which may be configurable by an administrator.

34.14.1 Operations

The tasks an administrator might have to do: Add, remove, list, edit, set defaults for…

34.14.2 Operands

The things he might have to do them to. Dimensions, Units, Vectors and their coordinate systems, input rules and representations, output rules and representations.

35 My Height: A Model For Numeric Information

(March 1993)

35.1 Introduction

Units of measure and data types have always been awkward to integrate into database schemas. It's especially tricky in object models, with their insistence on a clean separation between interface and implementation: on which side do units and data types belong? The problem surfaces yet again in dealing with domain mismatch when integrating heterogeneous data sources.

An elegant approach treats units and data types as distinct constructs in their own right, mapping in two stages from abstract magnitudes to communicable and recordable symbols.

35.2 The Problem

I am six feet or 72 inches tall. How should that be modeled in a database schema, object oriented or otherwise? Although different sorts of data models use different syntaxes, a property such as birthplace or height is always describable as some sort of mapping between kinds of things:

Birthplace: Person $\rightarrow$ City,
Height: Person $\rightarrow$????.

What exactly does height map to? To begin with, do we mean 6 feet (an integer) or 6.0 feet (a "real")?

Height: Person $\rightarrow$ Integer?
Height: Person $\rightarrow$ Real?

Are Real and Integer types in the same sense as Person and City? What is the relationship between Real and Integer data types? In type graphs, we tend to show Real and Integer as disjoint types, having no instances in common. But in the "real" world of mathematics, integers are a subset of the reals, i.e., the number 6 is both an integer and a real number. Do we really believe that the integer 6 and the real number 6 are different things? Are we confused between different numbers and different ways of representing numbers?

We're also confused about the names of our data types. "Real" is a misnomer; at best we might be talking about the rational numbers, not all the reals. To be more precise, we are talking about a subset of the rational numbers expressible with a finite and usually fixed precision. Some languages call these

Floating Point, or Decimal, although Decimal sometimes means integers represented in the base ten, rather than two or eight or sixteen. (The actual set of data types isn't what we want to focus on; we'll pick some arbitrary set for discussion purposes.)

Beyond that, what do we do about feet and inches and meters? What is the type of the value of the height property? We might introduce types like Feet and Meters, or Height_in_Feet and Height_in_Meters …

Height: Person → Feet?
Height: Person → Meters?
Height: Person → Height_in_Feet?
Height: Person → Height_in_Meters?

If the height of people is expressed in feet and the height of buildings is expressed in meters, do they or don't they have the same sort of values for the height property? In some sense, we'd like to say the height of anything is a distance.

Where would we put such types in a type graph? Sometimes we put them in as subtypes of the numeric types. Does that make sense? Are heights and weights and ages numbers? Or are they some other concepts, whose measurements are expressed in numbers? Distance is essentially a space between two points, not a number. Weight is essentially a heaviness, not a number.

35.3 A Three-Stage Process

Abstractly, we can describe a simple three-stage process to deal with all this. In the first stage, the value of a property such as my height is simply a distance, i.e., a space between two points. Picture the answer being given by someone with arm extended, palm down, saying "So high." No joke. That is the direct answer to the question of how tall I am.

The second stage is measurement, mapping this into numbers. Units of measure are different ways of mapping such things into numbers, yielding different numbers for different units. The numbers are still abstractions, not represented in any particular data type. If we ask someone my height in feet, imagine him holding up all the fingers of one hand and the index finger of the other: "This many." If we ask my height in inches, you'll have to imagine seven people holding up all their fingers, with yet another person holding up his thumbs, all saying "This many".

The third stage is representation. The number of feet in my height would be written as 6 in decimal, VI in Roman, 110 in binary, 111111 in unary, six in English, seix in Spanish, and so on. (I'm ignoring quoting conventions; that's another level of complexity, which we don't need to tackle here.)

We're trying to discuss concepts, but we can't communicate without using some form of representation. So let's say that, for the moment, /—/ represents an abstract distance corresponding to my height, while * represents an abstract number corresponding to the number of points in a snowflake. The three stages can then be summarized as

Height(Bill) = /—/
Feet(/—/) = *
Decimal(*) = 6.

We can talk about Height(Person), or even Height(Bill), as an abstraction. That's okay so long as we only want to *think* about my height, as an abstract concept. As soon as we want to communicate it or record it, we have to make it more concrete and put it into some form of representation, requiring an intermediate stage of measurement.

Imagine the following dialog with a somewhat perverse computer:

You: Do you know Bill's height?

Computer: Yes.

You: Well, would you show it to me, please?

Computer: I can't. The screen's not big enough to display two points such that the distance between them is the same as Bill's height.

You: Listen, all I want is a number.

Computer: What number? Bill's height is a distance, not a number.

You: Would you please measure Bill's height!

Computer: Certainly. What units shall I measure it in?

You: Feet will do nicely, thank you.

Computer: All right.

You: Well, could you please show me the answer?

Computer: Sorry, I don't have any fingers to hold up. I can only display character graphics on your screen. In what sort of displayable symbol would you like the number represented?

You: A decimal integer, for Pete's sake.

Computer: Sorry, I'll try to remember that that's what you expect next time. The answer is 6.

You: Well, thanks a lot.

35.4 Measurements And Data Types

Let's use "magnitude" to mean a measurable abstraction such as distance, weight, chunks of time, and so on, before it gets measured or represented. Let's also agree that "number" means an abstract quantity, not expressed in any particular notation. Then we have:

$$\text{magnitude} \xrightarrow{\text{measure}} \text{number} \xrightarrow{\text{datatype}} \text{symbol}$$

Units of measure are functions that map magnitudes into numbers:

Feet: Distance → Number
Inches: Distance → Number
Meters: Distance → Number
Miles: Distance → Number

Pounds: Weight → Number
Grams: Weight → Number
Kilograms: Weight → Number

Quarts: Volume → Number
Gallons: Volume → Number

Degrees: Angle → Number
Radians: Angle → Number

…

Thus, for a given distance /—/, the mappings

Feet(/—/)
Inches(/—/)
Meters(/—/)

each yield a different (abstract) number.

Data types, in turn, are functions that map abstract numbers into character string symbols. If % denotes the number of inches in my height, then we might have

DecimalType(%) = 72,
OctalType(%) = 110,
HexType(%) = 48.

A data type has two aspects: a set of symbols and a mapping from numbers into that set. The set of symbols is typically defined as the finite sequences of characters from a specified character set. Hexadecimal symbols are defined over the set of characters {0-9,A-F}. The corresponding mapping is then defined as

HexType: Number $\rightarrow$ Hexadecimal.

Dates need similar clarification: are we talking about a certain day, or a particular way of representing it? My birthday is a certain particular day in the past, which has its own abstract existence in the same way as the number ten. That day is that day. There are lots of ways to represent it: February 19, 1936; 2/19/36; 19-Feb-36; 19360219; 50/1936 (the 50th day of 1936); the 12,803rd day in the 20th century; etc., etc., etc. Each of those corresponds to a different data type. We don't think of them as being different entity types, different kinds of things. They are just different representations of the same abstract concept.

As a matter of fact, we even have a kind of measurement concept for dates. Before we can represent a date, we have to pick a calendar in which to "measure" it. Dates come out quite different in the Gregorian, Chinese, Jewish, and other calendars, not to mention the various forms used inside computers. Each uses a different sort of "yardstick", as well as a different origin as reference point. Our calendar "measure" is analogous to an odd sort of yardstick, as though the three feet in a yard each had their own name, and each contained its own number of inches. So our year is divided into twelve months, each being named, and each having its own number of days.

Thus we again have a two stage process: pick a calendar in which to "measure" a day, then pick a representation format for the date within that calendar. Keeping this date analogy in mind might make it easier to understand the treatment of other numeric quantities.

35.5 Three Levels Of Abstraction

We can draw a fine line between concept and realization.

Suppose you ask me if I know how tall I am. I say yes; an idea forms in my mind, capturing in an indescribable way some notion of a particular degree of "bigness". But in order to *tell* you how tall I am, I ultimately have to transform that notion into something very concrete, involving very real matter and energy, which will have some impact on your eyes, ears, muscles, or other physical receptors. The same is true if I want to record that information somewhere, so you can get at it later on. Information has to be converted to very tangible configurations of matter and energy.

Symbols are an intermediate stage of that transformation from concept to material communication. Symbols make it possible for me to tell you how tall I am without having to scratch a mark on the ground as long as my height, and to tell you how much I weigh without having to hand you a rock as heavy as I am. Symbols are at the boundary between hardware and software. Software deals with computer systems as symbol manipulators; hardware implements symbols in the electrons, holes, and magnetic fields in silicon, copper, phosphors, etc.

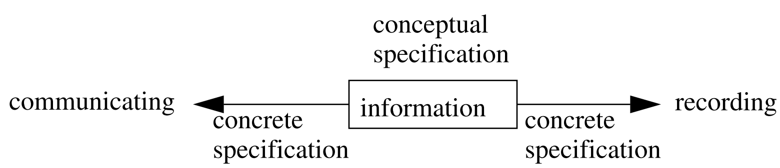
Symbols are also at the interfaces of computers. For our purposes, a symbol is anything that can pass across a communication or storage interface of a computer. Technological advances are rapidly expanding the domain of such symbols to include sounds, pictures, cursor movements, finger touches, hand-drawn script, and so on. For our immediate purposes, we can concentrate on symbols formed as linear sequences of characters. Such symbols can represent elementary concepts, aggregates that organize the elementary concepts into useful information, and procedures to be executed by the computer in processing information. This paper focuses on symbols representing elementary concepts.

Of these three levels of abstraction — conceptual, symbolic, and material — only the first two are relevant to data modeling. We'll say that conceptual specifications describe information at the conceptual level, while concrete specifications describe the symbolic level (admittedly an abuse of metaphor, but the alliteration is attractive).

35.6 Conceptual And Concrete Specifications

We are now in a better position to differentiate between conceptual and concrete information specifications. This applies to any sort of data model, object-oriented or not.

Conceptual specifications describe the information to be maintained inside the computing system. Concrete specifications tell how to map that information into symbols which can be recorded, or which can be communicated between the computer and users. Note that concrete specifications apply to two facets: how the computer communicates at interfaces with users, and how the computer records information internally in storage media:



For our purposes, the communication side includes delivery of information into variables in an application's program space, and also the encoding of parameters for requests to be sent to the information system. Thus a *conceptual* specification might define operations for getting and setting the height of a person, or for moving a vertex, as

Height: Person $\rightarrow$ Distance
Height: Person $\leftarrow$ Distance
Move: Vertex, Distance, Direction $\rightarrow$ Location

without specifying units or data types for any of the parameters. (That syntax is illustrative only, and should not provoke discussion of the semantics of attributes or update.) In an object model, that might be the form of an abstract object interface.

Conceptual specifications suffice to describe algorithms, procedures, and other relationships in the information. For example, operations can be specified to compare heights, or to add and subtract them, without reference to representation in terms of units or data types. The expression

Height(Bill) - Height(George)

is a meaningful operation on distances, not numbers. The result is an abstract distance, i.e., a space between two points (the tops of our heads? — there's a joke there).

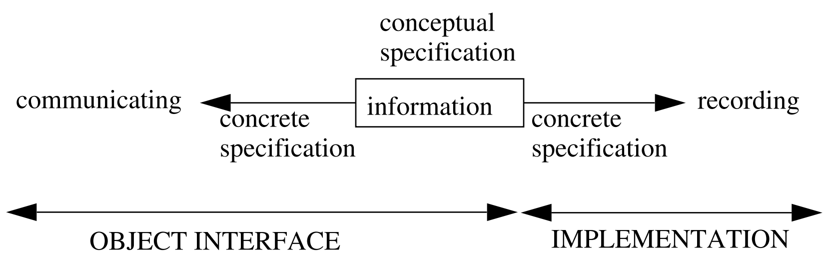
Concrete specifications, augmenting conceptual specifications with units and data types, would define external interfaces, which might be characterized as client or application or presentation or communication interfaces.

This configuration corresponds nicely with the ANSI three-schema architecture 9.. The client interface is the external view, the conceptual specification is the conceptual view, and the storage specification is the internal view. Both the client and storage specifications are concrete.

Data independence arises from the separability of the concrete specifications for external and storage interfaces, being linked via the conceptual specifications. A given conceptual specification can be implemented in different storage specifications, involving different units and data types (as well as different data structures and program code). These differences might arise serially, as when an application is ported to different environments or underlying implementations are tuned for performance. Different implementations might also be experienced concurrently, as in multi-database integration. There can also be different external specifications, involving different units and data types, as well as different display structures, e.g., tables, pie charts, and bar charts.

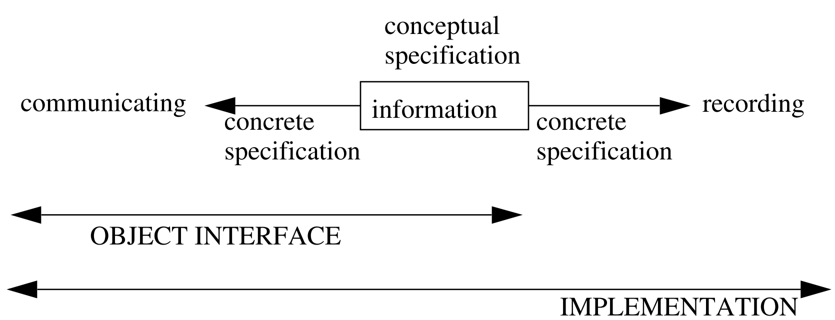
In practice, an "access path" can be compiled for a particular pair of external and storage specifications, allowing for efficient execution of a single composite mapping.

This configuration contrasts with current object models, which lump the client interface and the conceptual specification into a single notion of object interface. The storage specification then corresponds to implementation.



Though the partitioning is correct, this description is oversimplified by neglecting data structures and procedures.

It might be more accurate to describe current object models as.



That is, an object interface includes the client's concrete interface, and an implementation includes a description of the object interface it implements. This configuration allows only one client interface, but many storage interfaces, for a given conceptual specification.

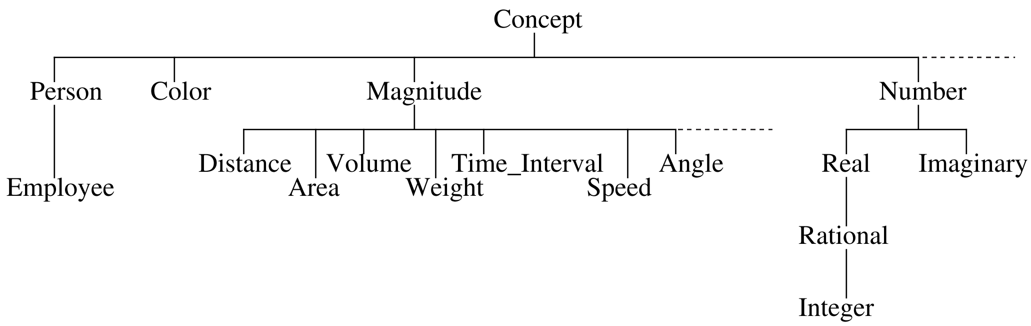
35.7 Conceptual Specifications

Conceptual specifications describe the categories of things occurring in the information, and the associations defined among those categories of things.

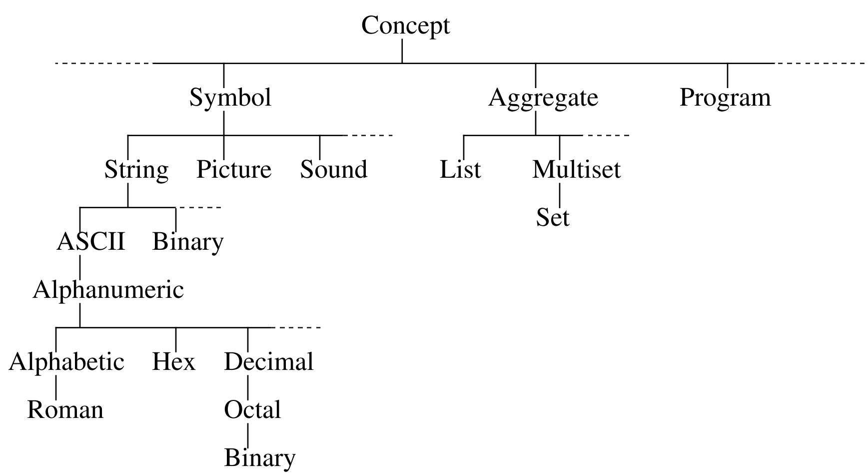
From the conceptual point of view, magnitudes such as distances, weights, time intervals, velocities, angles, etc., are all very distinct kinds of things, and all distinct from the notions of numbers. This is in keeping with a fundamental principle that distinguishes object models from value-based models: things have an existence and identity apart from the values of their properties. People exist independently of their social security numbers or employee numbers. Colors exist independently of the names we might call them. So then does a certain distance, or a certain degree of heaviness, exist apart from the units in which they are measured. So also do numbers exist apart from the symbols we use to represent them 7..

Our ontology of concepts includes such categories of things as persons, employees, colors, distances, time intervals, velocities, angles, real and imaginary numbers, rational numbers, integers, and so on. We know some relationships among these categories: employees constitute a subset of persons, and the integers are a subset of the rationals which in turn are a subset of the reals.

A diagram of this conceptual ontology might take the form



Symbols are themselves concepts, since we can think about character strings and numerals. For illustrative purposes, we might subdivide symbols into such things as images, sounds, and strings over various character sets. Aggregates such as multisets, sets, and lists are also abstract concepts. So also are programs, a generic term for various concepts expressing behavior. Thus the ontology might be extended with



The strings are sets of characters drawn from various character sets. Thus Hexadecimal contains strings over the set {0-9,A-F}.

Associations (mappings, functions, perhaps even operations) are defined in terms of the categories of things they associate. (In this paper we neglect further specifications of the criteria or algorithms which determine such associations, such as procedure bodies, constraints, or pre- and post-conditions.)

The concept of height is an association between persons and distances:

Height: Person $\rightarrow$ Distance.

The concept of name is an association between persons and alphabetic strings:

Name: Person $\rightarrow$ Alphabetic.

An operation for moving a vertex would also be defined here:

Move: Vertex, Distance, Direction $\rightarrow$ Location.

At this conceptual level, we also understand the concepts of measurement and representation. Thus the following sorts of associations would also be in the conceptual specifications:

Feet: Distance $\rightarrow$ Number
Inches: Distance $\rightarrow$ Number
Meters: Distance $\rightarrow$ Number
Miles: Distance $\rightarrow$ Number
Pounds: Weight $\rightarrow$ Number
Grams: Weight $\rightarrow$ Number
Kilograms: Weight $\rightarrow$ Number
Quarts: Volume $\rightarrow$ Number
Gallons: Volume $\rightarrow$ Number
Degrees: Angle $\rightarrow$ Number
Radians: Angle $\rightarrow$ Number
…

HexType: Number $\rightarrow$ Hexadecimal
OctalType: Number $\rightarrow$ Octal
…

That is, the notions of measuring magnitudes and representing numbers are understood at the conceptual level. What isn't specified at the conceptual level is how such things as the heights of people are measured or represented.

35.8 Concrete Specifications

The conceptual specification treats the computer as a black box full of information. We don't yet know how it records or communicates the information. To design mechanisms for recording or communicating, the conceptual specification needs to be augmented with concrete specifications mapping all information into symbols. Otherwise we don't know how to tell the computer how tall I am, how the computer will remember that, or how the computer will pass that information on.

Given a fact abstractly specified as

Height: Person $\rightarrow$ Distance,

the description of how to implement this fact in stored data, or how to communicate it in response to an inquiry, needs to be augmented in some form such as

Height: Person $\rightarrow$ Binary(Feet(Distance)),

or

Height: Person $\rightarrow$ Decimal(Inches(Distance)).

As a syntactic device to emphasize the separation of conceptual and concrete specifications, we could employ a notation such as

Height: Person $\rightarrow$ Distance $\Rightarrow$Feet $\Rightarrow$Binary,
Name: Person $\rightarrow$ Alphabetic.

A vertically aligned notation might make it easier to map multiple parameters:

```
Move: Vertex, Distance, Direction    → Location
        |        |          |           |
        |        cm       degrees    rect_coord = <distance, distance>
        |        |                                  |          |
       oid    decimal                               cm         cm
                                                    |          |
                                                 decimal    decimal
```

In that format, it might also be easier to distinguish conceptual and concrete portions of the schema, since they occur on separate lines. Different concrete specifications (implementations) can be obtained simply by replacing the “concrete” lines of the specification. (More elaborate implementation specifications would include data structures and method code. That could be addressed in another paper.)

The mapping to concrete specification is not always a simple two-stage mapping. Clearly if the information is already symbolic, as with names, then no mapping is required at all. If the information is already numeric, as with counts (e.g., inventories) or ratios, then no measurement is involved, but data type mapping is still required.

Complex magnitudes, on the other hand, might involve more than two levels of mappings. Locations in two dimensions might be measured either in rectangular or polar coordinates (first level of mapping). For either of these, the distances and angles involved themselves go through two more mappings for measurement and representation.

There is potentially yet another level of mapping, for such things as edit formats or masks, e.g.,

- Money: currency symbol, periods and commas, leading blanks and zeros.
- SS Nums: embedded dashes.
- Dates: various format embellishments.
- Names: First, Last, Initials.

These might be handled as a specialization of data types, but probably better handled as a separate level of mapping. These are symbol-to-symbol maps.

With conventional documentation techniques, it's difficult to maintain consistent replicas of the conceptual specs together with each different set of concrete specs in which it is realized. (E.g., class definitions are sometimes said to incorporate type definitions.) Modern media management techniques can alleviate this, allowing a single “master copy” of the conceptual specification to be overlaid with different concrete specifications.

35.9 Real Computations

The units and datatype functions described above are hypothetical abstractions, fictions and figments of our imaginations. They might be executed in our minds, but never in any real computers. Real computations are mappings among symbols, not among magnitudes or abstract numbers. There is no real operation that converts a height to an abstract number, or converts an abstract number to its decimal representation.

Instead of datatype mappings between numbers and symbols, we have type conversions between symbols in the compiled composite mappings between external and internal specifications. In principle, the conversion from hex to decimal involves mapping a hex symbol to the abstract number it represents, and then to the decimal representation of that number:

symbol(in hex) → number → symbol(in decimal).

In practice, of course, there's only one computation, directly from one symbol to the other:

symbol(in hex) $\rightarrow$ symbol(in decimal).

The same is true of units conversions. A conversion from feet to meters never materializes an actual abstract distance as an intermediate result. The conversion first accounts for data type differences, and then maps from the representation in feet to the representation in meters.

Real requests typically involve a chain of computations beginning and ending with symbols: input data, output data, or stored data. Intermediate results can be opaque (not specifying units or data types), but the computation is ultimately governed by a pair of representations at the beginning and the end. That determines the conversions required; the intermediate mappings to magnitudes and abstract numbers need never be visibly (concretely) executed.

35.10 Blurring The Distinction With Precision

If I'm six feet tall, then I'm also 1.8288036 meters tall — more or less.

Numbers map into symbols imperfectly. Communication and storage media can't handle infinitely long symbols. They work best with symbols of bounded length, limiting the precision with which numbers can be represented. Thus it might only be possible to store or communicate integers between 0 and 2^{31}.

Constrained precision blurs the clean line we try to draw between conceptual and concrete specifications. Sometimes that's handled by implicit conventions; while the conceptual specification may say integers are supported, everyone “knows” certain integers are too big to handle. This may be spelled out as an implementation restriction in a software manual, or it may be “common knowledge” based on the word length of the underlying computer.

A certain amount of respectability is gained by abstractly defining finite subsets of numbers, such as short and long integers, or 31-bit integers, in signed and unsigned variants. These at least have the merit of defining the populations abstractly in the conceptual specification, even though they are induced by underlying implementation constraints.

Such compromises are a fact of life we endure all the time. You can't hide implementation completely. It keeps sneaking in between the cracks, nudging us with “implementation restrictions”. As hard as our calculators and computers try to implement the model of arithmetic, the best they can do is finite calculation. Truncation and roundoff keep intruding non-arithmetic behavior: 3*(1/3) comes out 0.9999999999, not equal to 1. We don't alter our concept of arithmetic because of that.

35.11 Blurring The Distinction With A Need To Know

Sometimes a client interface will not specify a desired unit of measure or data type. This may be because it is implicitly assumed that the stored data satisfies the client's needs (data dependence), or because the client wishes to accept the data in whatever form it is stored to avoid conversions. In the latter case the concrete specification may include the units and/or data types as data items, making the mappings more complex. In effect, the concrete specification may take the form

Height: Person $\rightarrow$ Symbol x Measure x Datatype,

where the Measure and Datatype information might have to map to something in the mapping between the conceptual and storage specifications. This more complex situation bears further investigation.

35.12 Curried Equivalences: Alternative Models

The essential point is to recognize magnitudes, units of measure, numbers, data types, and symbols as distinct constructs. Measures and data types don't have to be modeled as functions.

Alternative models can be developed in terms of curried equivalences, which define a kind of equivalence transformation which can exist at the schema or model (meta-schema) level. At the schema level, it accounts for one sort of schema mismatch in integrating heterogeneous databases 8.. At the model level, it can define some equivalences among different models.

The general idea is that, given a function

$f: X, Y \rightarrow Z,$

there exists a set of functions

$f_i: Y \rightarrow Z,$

one for each member x_i of X, such that

$f_i(y) = f(x_i, y).$

Conversely, given a set of functions having similar signatures

$f_i: Y \rightarrow Z,$

there is a corresponding set of objects X containing one x_i for each f_i, and a function

$f: X, Y \rightarrow Z$

such that

$f(x_i, y) = f_i(y).$

We will make use of the converse transformation. Let's relabel the units of measure discussed above as InchesF, FeetF, MetersF, etc., to emphasize that they are functions:

InchesF: Distance → Number
FeetF: Distance → Number
MetersF: Distance → Number.

Those are the f_i, with Y being Distance. For X, we can introduce a set of Units whose members x_i are InchesU, FeetU, MetersU, etc. For f, we introduce a new function Measure:

Measure: Units, Distance → Number

such that

Measure(InchesU,d) = InchesF(d)
Measure(FeetU, d) = FeetF (d)
Measure(MetersU,d) = MetersF(d)

Conceptually, this formulation makes the process of measurement an explicitly visible activity, with the units of measure being passive participants (parameters). It is also more conducive to allowing units of measure to be stored with self-describing information. This latter might arguably be a more satisfying intuitive treatment, but there is nonetheless an equivalence mapping into the model developed earlier.

Data types can be treated similarly, using Represent as the collective function:

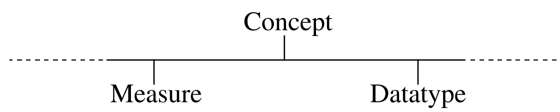
Datatype = {Integer, Hex, Octal, …}

Integer: Number → Symbol
Hex: Number → Symbol
…

Represent: Datatype, Number → Symbol

Represent(Integer,n) = Integer(n)
Represent(Hex,n) = Hex(n)
…

In this approach, the graph of conceptual categories is extended to include measures and data types:



The two approaches can be unified if we don't mind letting functions also be things which themselves can occur as arguments to other functions. The members of Measure and Datatype could themselves be functions, such that InchesU is in fact InchesF. Then Measure and InchesF could both be functions, with the equivalence

Measure(InchesF,d) = InchesF(d)

35.13 Conclusions

Units of measure and data types are distinct constructs which map magnitudes into symbols. This approach eliminates a lot of confusion in conceptual information specifications, whether in the context of the ANSI three-schema architecture or in the context of abstract object interface specifications. The approach also facilitates a systematic treatment of domain mismatch for multi-database integration.

Magnitudes such as distance, weight, and time are distinct concepts, distinct from the numbers that measure them and from the symbols that represent those numbers. They are related to the properties of objects by all or some of the following mappings:

$$\text{object} \xrightarrow{\text{property}} \text{magnitude} \xrightarrow{\text{measure}} \text{number} \xrightarrow{\text{datatype}} \text{symbol}$$

In conceptual specifications, information involving magnitudes is expressed purely in terms of those magnitudes:

$$\text{object} \xrightarrow{\text{property}} \text{magnitude}$$

e.g., Height: Person $\rightarrow$Distance.

Concrete specifications are required to map these into symbols both externally for communication with clients and internally for recording in storage implementations. Concrete specifications use units of measure and data types to map magnitudes into symbols:

$$\text{magnitude} \xrightarrow{\text{measure}} \text{number} \xrightarrow{\text{datatype}} \text{symbol}$$

Data independence is obtained through the independent mappings of external and internal interfaces to the conceptual model. Current object models already separate object interfaces from implementation. One further distinction is required, separating the abstract object interface from client interfaces.

Domain mismatch in multi-database integration can be handled in similar steps. Correspondence between attributes should first be established at the conceptual level in terms of common magnitudes, e.g., distance or weight or time, etc. Then the units of measure and data types used to implement these can be identified, with appropriate conversions specified.

35.14 References

7. William Kent, “A Rigorous Model of Object Reference, Identity, and Existence”, Journal of Object-Oriented Programming 4(3) June 1991 pp. 28-38.

8. William Kent, “Solving Domain Mismatch and Schema Mismatch Problems With an Object-Oriented Database Programming Language”, Proc. 17th Intl. Conf. on Very Large Data Bases, Sept. 3-6, 1991, Barcelona, Spain.

9. D. Tsichritzis and A. Klug (eds), “The ANSI/X3/SPARC DBMS Framework. Report of Study Group on Data Base Management Systems”, AFIPS Press, Montvale NJ, 1977.

36 The Grass-roots Basic Research Program Project

Rezal Rahmen (Stanford University) assisted us on this project in the summer of 1995 under a grant awarded by the Hewlett-Packard Laboratories Grass-roots Basic Research Program (GBRP). Following are the initial project proposal and the slides of the final presentation.

36.1 The Proposal

GBRP PROPOSAL – DIMENSIONED DATA: BEYOND DIMENSIONAL ANALYSIS

Investigators:

Bill Kent, HPL/CRC/STL/AED Advanced Object Practices project
Stephanie Leichner, CSO/SBU/SESD
Bruce Hamilton, HPL/MRC/IPL Measurement Systems Dept.
Dan Hepner, CSO/CSG/GSY/GSSL High Availability Core project

This proposal begins by describing a larger research context, and ends by defining a small subset as a candidate GBRP for this year.

While measurement constitutes 1/3 of MC^2, HP has no coherent theory of measurement, and no scientific research into its foundations. Traditional dimensional analysis is too limiting.

For example, fuel consumption in gallons per mile is a volume divided by a length, yielding length^2, which is area. Traditional dimensional analysis therefore considers fuel consumption equivalent to area, meaning that the two can be freely added and compared, and one can be assigned to the other. Is adding fuel consumption to area any more sensible than adding area to volume? Similar anomalies arise with paint coverage (gallons/square foot) and length; licorice productivity and speed (miles/hour); and work and torque (both are force * distance, yet they are not equivalent).

There are a host of dimensions which are “dimensionally equivalent” without being semantically equivalent. They aren’t compatible with each other, even though dimensional analysis says they are. A potential solution under investigation takes the form of “specialized dimensions”. Speed and linear productivity could be defined as distinct specializations of length/time. Making them distinct renders them incompatible, while recognizing a common parent lets them inherit common units.

The opposite problem exists, too. There are dimensions which are semantically equivalent, and should be compatible, even though they aren’t dimensionally equivalent.

If I have a cup of salt and you have a pound of salt, it makes sense to ask who has more salt, and we could figure out how much salt we would have if we added mine to yours. But dimensional analysis wouldn’t allow us to compare or add a volume and a mass, even though both are notions of “amount”. Similar things could be said about concentrations and unit costs.

Mass and volume can be converted once the density of the substance involved is fixed. The approach being considered here is to introduce a “quasi-dimension” such as “amount”, defined as a generalization of such things as mass and volume (and perhaps length or area, if the subject is rope or carpeting). Once the conversion parameters are fixed for a given computational context, the members of a quasi-dimension can be converted in much the same way that the units for a given dimension can be converted. Beyond “amount”, other quasi-dimensions include such things as “concentration”, “unit price", and "travel distance" (often expressed as either distance or time, as on AAA maps; astronomical distances also are expressed as either distance or time). This has important applications in many industrial settings, and in the medical context, where medication dosages are routinely converted between different dimensional forms.

There are other sorts of theoretical concerns as well:

- Dimensional analysis treats dimensionless quantities such as shrinkage (length/length), clock accuracy (time/time), and angle (also defined as length/length) as being dimensionally equivalent.
- Support for dimensioned data in programming languages should be extensible, allowing the definition of new units and dimensions (e.g., money). What are the criteria for determining whether a proposed measurement domain fits the computational model provided for dimensioned data? Why or why not will such a system be able to support measurement of color, performance, usability, complexity, velocity, and others? The decision would be facilitated by a set of axioms describing the fundamental behavior of measurement domains, against which any candidate domain can be tested.
- Arithmetic behaves differently in different measurement domains. 40 degrees C is sometimes equivalent to 104 degrees F and sometimes to 72 degrees F. An angle of 400 degrees is sometimes equal to 40 degrees, sometimes not. Subtracting a larger length from a smaller one is sometimes legal, sometimes not. Extension of the axiomatic system mentioned above, as well as introduction of yet other relationships among dimensions, would explain some of these behaviors.
- An underlying semantic model which clearly differentiates physical quantities, units, and representations provides a coherent basis for the definition of computational support.

These concerns will not be elaborated in this brief proposal. They are part of longer ongoing work. This work has been in progress for some time on a voluntary basis, outside the scope of current projects. Potential deliverables:

- Specifications for type system extensions and computational models in programming languages. It could become a class library for object-oriented languages.
- Enhancements to application analysis and development methodologies.
- Contribution to scientific knowledge, extending the theory of dimensional analysis.

Either or both of the following two tasks would be a feasible GBRP project. Both have to do with specialized dimensions and quasi-dimensions as described earlier:

1. Develop a test suite and validate the approach for a number of interesting cases.

2. Develop specifications and language bindings in one or more languages for the implementation of these features.

A summer student for this project should be competent in programming language type systems.

36.2 The Final Presentation

Dimensioned Data Investigation

Background:

Started in 1993.

Rafiul Ahad (CSY) sought support for dimensioned data in OpenODB for POSC.

Continuing now as an unfunded background effort.

Present investigators:

Bill Kent, HPL/CRC/STL/AED DADO project

Stephanie Leichner, CSO/SBU/SESD

Bruce Hamilton, HPL/MRC/IPL Measurement Systems Dept.

Dan Hepner, CSO/CSG/GSY/GSSL High Availability Core proj.

Status:

In progress. 125-page workbook.

Focus

Fundamental semantics.

Beyond traditional problems of units conversion, precision, accuracy, etc. (But they still need attention.)

Emerging themes:

- **Basic model for supporting dimensional computation.**
- **Anomalous behaviors within dimensional analysis...**
  - **Specialization**
  - **Generalization**
  - **Siblings**
- **Axiomatic foundation.**
- **Non-simple structures: vectors, aggregates.**
- **Skeletons in the closet.**

Basic Model of Dimensional Analysis Support

Fundamental principles:

A measured quantity is a distinct entity from the various measurement expressions that represent it.

Treat dimensions as types.

Anomalies

Incompatible Dimensions Look Compatible

Work and Torque both have the form Force x Distance (ML^2/T^2).

Fuel Consumption (gallons per mile) is Volume/Length, i.e., Area.

Paint Coverage (gallons/sq ft) is Volume/Area, i.e., Length.

They shouldn't be compared or added.

But they are compatible under dimensional analysis.

Solution: Specialized dimensions.

Subject of the GBRP summer investigation.

Anomalies

Compatible Dimensions Look Incompatible

If I have a cup of salt and you have a pound of salt, who has more?

You and I can figure that out, but dimensional analysis can't handle it. Volume and weight are incompatible.

Solution: Generalized dimensions.
(Substance dependent, of course.)

Important applications in medicine, industry.

Computational Anomalies

Temperature: 0°C is 32°F or 0°F?

Angles: is 400° the same as 40°?

Weights: is (5 lbs - 10 lbs) a legitimate weight?

Solution: differentiated dimensions:

- **Temperature point vs. temperature interval.**
- **Rotational angle vs. circular angle.**
- **Weight vs. buoyancy.**

Axiomatic Foundations

Purpose:

- **Explain behaviors of differentiated dimensional domains.**
- **Criteria for determining whether and how new dimensions can be supported.**

    Money, color, performance, efficiency, usability, ...

Basic approach: characterize the behaviors of measured quantities without assuming they behave like numbers.
Justify modeling them with numbers.

What is Specialization

- **Specialization involves creating a type hierarchy for dimensions that share a common base dimension.**
- **For example, Energy, Work, and Torque share the common base dimension**

    mass x length^2 x time^{-2}

Why Specialize?

- **Distinguishing dimensions that share a common parent.**
- **Error detection during compilation.**

Without Dimensional Analysis:

```
boolean: b;
real: A, l, w, wt;
/* area, length, width, weight */
A := l*b; <---- error caught
A := l*w; <---- ok
A := l*wt; <---- error undetected
```

Ordinary Dimensional Analysis:

```
length^2: A;
length: l, pc; /* paint coverage gals/sq ft */
A := l*wt; <---- error caught
A := l*pc; <---- error undetected
```

Specialization:

```
length: l;
length^2: A;
paint_coverage: pc;
A := l*pc;
```

But there are problems...

Problems with Specialization

```
width: w;
height: h;
length: p; /* perimeter */
w := h; <---- undesired operation
p : = 2(w+h); <---- desired operation
```

Specialization Rules

- **Specialization rules are introduced to enable flexibility in specialization.**
- **Rules enable specialized dimensions to be added or subtracted, multiplied or divided, compared, and assigned to other specialized dimensions.**

Summary of Rules

Rule 1: Permit addition, subtraction or comparison among siblings.

Rule 2: Permit assignment among siblings.

Rule 3: Permit assignment from parent to child.

Rule 4: Permit assignment from child to parent.

Rule 5: Permit addition or subtraction between parent and child.

Rule 6: Permit comparison between parent and child.

Rule 7: Permit promotion prior to multiplication or division.

Note: Default is disabling all the rules above.

Conclusions re Specialization

- Specialization offers more refined type checking.
- Specialized dimensions are self-documenting.
- Degree of specialization depends on application.
- Specialization may be too restrictive but can be relaxed with rules.
- Deeper specialization tends to require more rules.
- Proposed rules are all useful at times.
- Scope of usefulness of rules is still unclear.

The Unary Dimension

A special case of specialization for dealing with dimensionless dimensions.

"Unary" is a dimensionless base dimension (the "null" base dimension).

Others are differentiated as different specializations of Unary:

- Angle (perhaps several)
- Clock rate error
- Strain
- Slope
- Some concentrations (e.g., volume/volume)
- Other ratios?

Generalization

Renders certain "incompatible" dimensions compatible:

- Amounts (weight, volume, length)
- Other families based on amount:
  - Concentration (amount/amount)
  - Price (money/amount)
  - Dosage (amount/time)
  - ...
- Travel intervals (time or distance)
- Pressure (psi or mm Hg)
- Astronomical distances (length or time)

Sibling Dimensions

The family of angles:

Rotational	Circular	Interior	Heading
360°>180°>0°	180°>0°=360°	360°=180°=0°	360°=0°≠ 180° x>y: error
280°+280°=560°	280°+280°=200°	280°+280°=160°	280°+280°: error

Axiomatic Foundations

Motivation

Why do computational paradigms work in some domains and not in others?

Why do they work differently in different domains?

How can we assess the applicability of our paradigms to new domains?

Axiomatic Foundations

Measurement Paradigms

- **Units-based**
- **Counting**
- **Formula**
- **Judgement**
- **Other?**

Axiomatic Foundations

Axioms for Units-Based Measurement

The attitude: *verify*, don't assume, that the domain behaves isomorphically to the real numbers:

- **Well-defined equality (and hence differentiability)**
- **Combinational closure**
- **Coverage**
- **Order**
- **Zero**

Others?

Complex Dimensions

- **Vectors**
- **Aggregates**

Skeletons in the Closet

An illustrative list of problems and anomalies that need to be dealt with. See the workbook.