A STUDY OF THE USE OF ABSTRACT TYPES FOR THE REPRESENTATION OF ENGINEERING UNITS IN INTEGRATION AND TEST APPLICATIONS

Charles S. Johnson

ABSTRACT

Physical quantities using various units of measurement can be well represented in Ada by the use of abstract types. Computation involving these quantities (electric potential, mass, volume) can also automatically invoke the computation and checking of some of the implicitly associable attributes of measurements. Quantities can be held internally in SI units, transparently to the user, with automatic conversion. Through dimensional analysis, the type of the derived quantity resulting from a computation is known, thereby allowing dynamic checks of the equations used. Through error analysis, the precision with which a quantity is measured can be correctly propagated into the result of a computation involving that quantity. The output of both measured and computed quantities can automatically be rounded to the correct significance, and labeled with the correct units.

The impact of the possible implementation of these techniques in integration and test applications is discussed. The overhead of computing and transporting measurement attributes is weighed against the advantages gained by their use. The construction of a run-time interpreter using physical quantities in equations can be aided by the dynamic equation checks provided by dimensional analysis. The overhead of responding to measured and computed system variables in real-time systems can be decreased in the case where only the significant changes in data values are responded to. The effects of higher levels of abstraction on the generation and maintenance of software used in integration and test applications are also discussed.

INTRODUCTION

Data abstraction should, in the near future, become the most important tool used in the Ada development of replacements to current systems functioning in the area of Integration and Test (I & T). This importance stems from the urgent need to maintain Test Procedure/ Test System Independence. This independence promotes both the reusability of Test Procedures and the possibility of modifying physical device information in the Test System, at run-time, without affecting procedures using logical access methods. This is necessary to decrease turn-around time due to modifications of the Test System/ Test Article hardware configurations.

BRIEF BACKGROUND

Kennedy Space Center/ Engineering Development/ Digital Electronics Engineering Division is in the process of prototyping distributed systems supporting I & T applications, particularly the Space Station Operations Language (SSOL) System, which is the I & T subset of the User Interface Language (UIL) for the Space Station. The discussions in this paper were developed from the results of systems designed and developed in Ada to demonstrate the feasibility of supporting the abstract data types used in I & T, specifically, engineering units. The Ada environment used was that of VAX Ada under VAX/VMS.

SYSTEM CONCEPT

There is a direct correlation between the effectiveness of computer systems and the fidelity with which objects in those systems simulate the behavior of the external phenomena that they are intended to represent. [1] The definition of objects is then akin to a simulation effort: complete with objectives outlining progress towards simulation goals, and constraints which limit the scope of the effort.

The goal of object definition for measurements and quantities used in Integration and Test applications is to create objects representing the physical quantities that are measured, tracking a magnitude for the quantity, and the type of quantity. The quantities (VOLTS, METERS/SECOND, PSI) should interact with other quantities in the same way that real physical phenomena do:

$$V = IR$$
$$PV = nRT$$

In other words, arithmetical operations should convert the quantities correctly into new quantities. Also it would be useful if the creation, input, and output of those quantities could be performed using any unit or scale of measure (length in METERS or MICRONS or CUBITS). It would be nice, as well, to know the precision with which a measurement was made, so that it can be determined if it represents a significant change from the last measurement. That precision, or measurement error, should propagate correctly during computation as well.

The objectives which mark progress towards these goals can be established. The quantities and units should be easy to define and use. The quantities should convert correctly upon input in different units. The quantities should convert correctly upon computation, and if the resultant quantity is of the incorrect type, an exception should be created, because the equation is incorrect (or the result type is wrong). Precision should be computed correctly for the different arithmetic operations. Finally, if the wrong units are selected for input or output, an exception should be generated.

The constraints which confine the scope of the effort can be defined. It is important that the support of the features of the system should not incur excessive system

processing or storage size overhead, because too much time and space costs money (space less than time these days). The resulting packages should not be too complex, relying instead on algorithms and structures that are just complex enough to create a useful result. Lastly, the development should be constrained against passing the point of diminishing returns. If a feature is difficult to implement and yields little in tangible results, it should be forgone.

PHYSICAL QUANTITIES

The tracking and converting of types of quantities is simply and efficiently done in computer programs by dimensional analysis. [2] This involves some fairly simple physics, for example, the average acceleration of a body can be computed by the equation:

$$\text{Average Acceleration } (m/s^2) = \frac{\text{Velocity Change } (m/s)}{\text{Time of Change } (s)}$$

which uses the units m = meters and s = seconds. The average force applied to the same body can be computed by:

$$\text{Average Force } (N) = \text{Mass of Body } (Kg) * \text{Average Acceleration } (m/s^2)$$

which uses the units N = newtons and Kg = kilograms as well as meters and seconds. What can be seen from combining the units of these equations is the following units equivalency:

$$\text{newtons } (N) \cong \frac{\text{kilograms } (Kg) * \text{meters } (m)}{\text{seconds squared } (s^2)}$$

The units like kilograms, meters and seconds are called base units, and the units like newtons are called derived units. These are all SI units, standardized by the ISO Resolution R1000 in 1969, and documented in the Le Systeme Internationale de'Unites (BIPM), but conversions exist for all other forms of units as well. If a matrix of derived units versus their base units is made, the dimensionality of derived units in their base units can be shown (Table F.4.4-1). The newtons unit shows a one in the kilograms column, a one in the meters column and a -2 in the seconds column, because seconds squared is reciprocal.

This dimensional analysis can be done for most units in any of the systems (English, CGS, etc.). Some units, however, are truly dimensionless. An example is the decibel and the Richter scale units, which are logarithms of ratios of units which cancel out. Some units just do not fit into dimensional analysis. AC circuit impedance equations do not cancel nicely, for instance, and AC units would probably have to be defined as dimensionless for those equations to correctly cancel. This simple dimensional analysis, as a whole, probably deals badly with sinusoidal phenomena.

TABLE F.4.4-1: PARTIAL TABLE OF UNIT DIMENSIONALITY

For each unit, the dimensionality is given versus each base unit from which it is derived (meter, kilogram, second, ampere, kelvin, candela & mol), along with the scale and offset required (1.0, 0.0 for SI derived units).

	Base Units							SI Conversion
Derived Units	m	Kg	s	A	K	cd	mol	Scale, Offset
newton:N	1	1	-2					1.0, 0.0
hertz:Hz			-1					1.0, 0.0
joule:J	2	1	-2					1.0, 0.0
watt:W	2	1	-3					1.0, 0.0
volt:V	2	1	-3	-1				1.0, 0.0
lumen:lm						1		1.0, 0.0
henry:H	2	1	-2	-2				1.0, 0.0
molarity:M	-3						1	1000.0, 0.0
astron. unit:AU	1							1.496E+11, 0.0
footpound:ft-lb	2	1	-2					1.356, 0.0
knot:kt	1		-1					0.5144, 0.0
slug		1						14.5939, 0.0
fahrenheit:^{o}F					1			0.5556, 255.37

The advantages of this method of tracking dimensions are mostly in verification of physical equations used in I & T applications. Even very complex equations involving many factors can be analyzed. During addition and subtraction operations, the two input quantities and one output quantity must be identical dimensionally. During multiplication the dimensions are added, and in division they are subtracted. If the result type doesn't match the computed dimensionality, it is an error. Dimensions can be stored as integers of range -20..20, and the overhead involved in integer arithmetic and compares is probably little.

The disadvantages are that it doesn't deal well with AC quantities and the like, which would require a complicated and unwieldy solution, yielding few tangible returns. Also, there are several correct dimensional solutions, any of which can be misapplied to a problem, with no detectable dimensional error (series/ parallel DC circuit equations).

MEASURED PHYSICAL QUANTITIES

The measurement of physical quantities always incurs a measurement error which can be assigned to the measured quantity at it's source, as it enters the system. This precision is key to any analysis of the significance of the measured quantity. If two sequential measurements of the same phenomena are obtained,

and their difference is less than the precision by which they are measured, then there is no significant difference, and the measurement is considered the same, experimentally. The scope of use for measurement precision would then be anywhere, in the system, after the conversion from raw data (counts) into engineering units. It should be noted here, that the measurement precision analysis discussed is different from the significant change analysis used in the front end processing of raw counts in the Launch Processing System (LPS), which is a digital process for raw data concentration to remove line jitter.

Error propagates (increases) as measurement values are combined by physical equations to yield resultant quantities. If the precision is an available attribute of the measured quantity, then the precision of all computed quantities can likewise be computed and carried along with the measurement. The computation of propagated error from the mathematical operations applied to quantities is shown in Table F.4.4-2. The relative-type error, on the right, looks like it will produce many occasions of division by zero, and is therefore not useful. The absolute-type of error, in the center, looks to produce a divide by zero only when the operation is a divide by zero (in error), and seems optimal.

For quantities introduced into the system without a measurement, such as constants, the precision input would be derived from the number of significant digits (+/- one half of the last digit).

TABLE F.4.4-2: FORMULAE FOR LIMITING ERROR

For the following mathematical functions: f(x, y); given that x and y are the exact values, a and b are their measured approximations, and the deltas for a and b are their limiting errors.

Type Of Function	Bounds For The Absolute Error	Bounds For The Relative Error
$x+y$	$\triangle a+\triangle b$	$(\triangle a+\triangle b) / \lvert a+b\rvert$
$x-y$	$\triangle a+\triangle b$	$(\triangle a+\triangle b) / \lvert a-b\rvert$
$x*y$	$\triangle a \lvert b\rvert + \triangle b \lvert a\rvert$	$\triangle a / \lvert a\rvert + \triangle b / \lvert b\rvert$
x/y	$(\triangle a \lvert b\rvert + \triangle b \lvert a\rvert) / b^2$	$\triangle a / \lvert a\rvert + \triangle b / \lvert b\rvert$
x^n	$\triangle a \lvert n a^{(n-1)}\rvert$	$\lvert n\rvert (\triangle a / \lvert a\rvert)$

DERIVED TYPES SOLUTION

The simple object-oriented approach to measured quantities would be to consider units to be classes of measurements, and to make them derived types of a base record type which would have

the components mentioned: measured value and precision, and dimensional values. Then the combinations of these types would be performed by defining, for example, a multiply function that takes inputs of AMPS and OHMS and makes VOLTS. To define the legal combinations for just a few types would be laborious, there are just too many relationships. The simple approach is too complex.

DISCRIMINANT TYPE SOLUTION

A solution for the representation of physical quantities using discriminant records is pointed to in Hilfinger [3]. It is not written exactly in Ada, though, for he presents a case for possible changes in the language. The record discriminants are the dimension values and the units scale factor, which would then prevent assignment of dissimilar units of the same dimensionality. For example, quantities in meters could not be assigned to quantities in feet, although the dimensionality is the same. Assignment of dissimilar constrained records is then accomplished by the overloading of the assignment operator ":=", with a function that re-scales the internal value to the new scale factor, and creates the correct and matching constraint values.

In current Ada, however, only discrete discriminants are legal, which disallows units scale factor as a discriminant (because it is a real type), and the ":=" operator cannot under any circumstances be overloaded. So it doesn't work in standard Ada.

An attempt can be made to standardize that approach, but there are some problems without the fixes to Ada. If the scale value were kept as a record component, instead of a discriminant, it will be modified upon assignment (not a constraint anymore). This negates the ability to keep scale in the quantity, and the quantity scaled as feet, instead of meters.

If the scale for an engineering unit alone is kept, then offset units, such as degrees fahrenheit (not aligned with absolute 0 ^{O}K) cannot be used.

There is also unnecessary run-time overhead to re-scaling every time a computation is made, and possible rounding error in the scale, which may drift. The rounding error in the scale is probably the reason why Ada doesn't allow it or any other real type to be used as a record discriminant.

CLOSELY-COUPLED DISCRIMINANT TYPES SOLUTION

A further redefinition of an object can be accomplished with differentiation [1], when the object definition has become too amorphous to simulate the target phenomena. Differentiation could be considered a fine structure definition technique for systems, whereas Object-Oriented Design or Functional Decomposition are gross structure definition techniques.

A separation of the object definition for physical quantities is made, into two closely-coupled objects, QUANT and

UNIT. QUANT is the measured quantity, and UNIT is one possible engineering unit for a quantity. They have attributes in common, the dimensional values. They also have unshared attributes.

UNIT is a private dimensionally-constrained discriminant record which contains the scale and offset for the engineering unit it represents, and can have for components other engineering unit attributes, such as the text label for output functions, or an input prompt text.

QUANT is a private dimensionally-constrained discriminant record which contains the measured value stored in SI units (no re-scaling), and can have for components other measurement attributes, such as measurement precision or identification of source device or process.

Arithmetic interactions of real types with type QUANT should be similar to those between scalars and vectors, only multiplication and division being allowed for scaling the QUANT values. Arithmetic interactions of real types with type UNIT should be similar to those between scalars and unit vectors, and therefore a QUANT is the outcome. Any arithmetic interaction of a UNIT with a UNIT or a QUANT should produce a QUANT, converting the pure to the impure, so to speak. QUANT objects should arithmetically combine to produce QUANT objects, of course.

With these definitions for the private types and arithmetic functions, it is simple to define several QUANT subtypes for the physical quantities (LENGTH, MASS, VELOCITY, POTENTIAL, WORK, INDUCTANCE, etc.) and to define several UNIT constants (deferred constants in the package) for the engineering units (FT, KG, KPH, VOLTS, FT_LB, HENRYS, etc.). It should be simple to create values for QUANT on the fly:

```
PIPE_LENGTH : LENGTH := 5 * FT;
GAS_CONSTANT : CONSTANT QUANT :=
                        8.31434 * JOULES / ( DEG_K * MOLS );
```

Functions for creating new UNIT constants on the fly will be necessary, since they cannot be produced arithmetically or defined externally to the package. I/O functions for QUANT values will also require a UNIT constant as a parameter, for scaling to/from SI units. A function for extracting the value of a QUANT object as a real variable, will also require a UNIT parameter and a conversion.

It would be possible, with a private dimensionally-constrained discriminant record variant, to create one type by lumping both QUANT and UNIT attributes together (one discriminant chooses which). This, however, is an unsatisfactory technique. With variant objects, the programmer always has to check what he has, before he can use it. The overhead of such checking is little, but the complication is now pushed into the application, instead of being in the package. This would seem to be a reversal of the purpose of abstraction.

Measurement precision could be included as a component in the QUANT definition by the use of the absolute precision computations listed in Table F.4.4-2. The absolute quantity

precision would then be computed into any result, like the dimensionality and the measured value itself.

The offset component of the UNIT type, could be used for more than just offset temperature scales (celsius, fahrenheit). Any differential scale could be represented by an engineering unit. In an example, cargo positional coordinates could be internally held in a centralized coordinate scheme. Differential voltages or pressure readings from sensors could also be related to some reference point.

If the pre-defined UNIT constants were ordered into a table, the I/O functions could, given a quantity of unknown type, select an output label and scaling, or an input prompt. This might be particularly useful in the generation of reports or ad hoc queries, which would use computation involving quantities and creating new quantities on the fly.

USE OF DISCRIMINANT TYPES IN GENERICS

Along with the useful constraining features of discriminant records, comes the difficulty of matching them with generic formal parameters. To instantiate a generic software component with a formal parameter matching a discriminant private type, the type must be constrained (no unconstrained types in generics), and the type of constraint passed as a generic formal parameter first. Then the discriminant type is passed, as a discriminant generic formal parameter. It is fairly obvious that most generic software will be produced, not of this type, but using the type private, with accompanying functions of that type (as in the generic sort function in most textbooks).

This problem can be handled, for any unconstrained discriminant type (QUANT, UNIT) with constrained subtypes (VOLTAGE, POWER), by declaring a non-discriminant record type which contains the unconstrained discriminant type. To instantiate a generic sort function, the enclosing record type would have to be passed to the generic, for the creation of an array type (for sorting), and an ordering function ">" for the enclosing type would also have to be defined and passed.

This is somewhat of a kludge, in that the constraints do not apply within the scope of the generic component. Simply put, the discriminant types feature of Ada somewhat precludes the use of generic software in a straightforward manner.

ANALYSIS OF OVERHEAD FOR USAGE OF OBJECT DEFINITION

The storage overhead can be estimated on the assumption of bytes for dimensional integers and 4-byte floating point representation for the measurement itself and for it's precision. This gives a 2X storage increase for carrying the dimensions and another 1X for the precision, up to 4X for everything. In communications, with all of the rest of the overhead involved in sending a measurement in a packet, this probably is not significant (other measurement information,

status, device status, send/receive addresses, transaction ID, packet ID, etc.).

The computational overhead for the dimensional analysis feature, which uses integers, is thought to be small compared to the floating point math involved in each multiply and divide for the measurement itself. This is thought even though there are seven dimensional integers being added for every measurement being multiplied (inversely for divides). Measurement adds and subtracts simply involve comparing for the dimensions (can't add VOLTS to WATTS).

The computational overhead for the precision feature, if absolute error is propagated by a floating point representation, is about 1X for adds, subtracts and the power function, 2X for multiplies, and 4X for divides. This can be seen in the central column of Table F.4.4-2.

ADVANTAGES OF DIMENSIONAL ANALYSIS FEATURE

The low computational overhead incurred by this feature is more than compensated by the advantages it carries. These are in the area of verification, validation, and run-time interpretation support.

During the development of I & T software, the use of constrained types to represent quantities should make possible the verification by dynamic analysis of that software. Even the most complex equations using dimensional variables, can be checked for the correct and allowable combination of subtypes, and for the return of the correct types of physical quantities. However, this will not catch those mistaken computations which return the correct quantity type, incorrectly computed.

The dimensional analysis method, since it is a dynamic feature of programs using it, and not a static feature, will lend itself well to validation of programs as well. In large I & T applications (for example LPS), the binding of logical measurement designation to physical device parameters is delayed for as long as it is possible. This allows the modification of hardware parameters with the minimum impact on the software system. The optimum circumstance would involve run-time binding of the logical level to the physical, so that the hardware configuration could be changed at test-time without having to patch the system, as is done now.

In that desired situation, there will be a large separation between the analysis of the logical nature of a program (equations), which would occur during development and verification, and the physical validation of the program against the model, or components of the Test System. As the distance between the verification of the logical and the validation of the logical-to-physical widens, the potential for dynamic problems to escape unnoticed should increase. If methods for logical verification of programs at run-time are used, such as dimensional analysis of equations by the method proposed, the possibility of catching these dynamic problems increases.

This problem of run-time dynamic analysis is exacerbated in the use of I & T command languages such as the User Interface

Language (UIL) and it's subset, the Space Station Operations Language (SSOL). The commands in these languages are interpreted at run-time, and are formulated by the user at the terminal, on-orbit. The use of complex equations in these languages to perform control functions is proposed. Syntax checking can be performed easily by the User Interface, but checks for correctness of the physical equations used will require some facility, such as dimensional analysis. If dimensional analysis were used, the internal checks in the interpreter program would be automatic against every statement using physical quantities, and exceptions would be generated on a statement by statement basis.

ADVANTAGES OF COMPUTATION OF PRECISION FEATURE

As the complexity of systems increases, leading up to the Space Station era, so does the number of levels of integration to be passed through by components before their operational use. In some shuttle payloads from ESA, there are already 7 levels of integration. If Ada is to become widespread in it's use as the I & T language supporting these levels of integration, then the Ada software products must be promotable between levels of integration. This does not imply a need to run the same procedure at a higher level, although that may be a requirement. What it does require is that lower level software components be incorporated by some method of abstraction into higher level components, up through launch operations and on-orbit operations.

At each level of abstraction, component level state, control and measurement variables are presented as parameters to higher-level integration software, simplifying interfaces at the subsystem, and then the system level. This continues until at the on-orbit user interface level, simple designators for systems are connected to a large tree of state, control and measurement variables extending all the way back down the integration chain.

In systems using abstracted measurement variables, knowing the significance of measurements at all levels is an important issue. Control logic algorithms which attempt to establish set points to an insignificant range are erroneous. Commands which effect an insignificant change in an effector are meaningless, and consume system resources in their performance. Measurements which involve insignificant changes in levels should not be communicated.

Communication overhead becomes more of an issue, as we progress from tightly-coupled shared-memory systems (like LPS), to loosely-coupled distributed systems (like GDMS prototypes). Distributed systems have failure modes related to communication loading (traffic jams), which can be abated somewhat by data concentration.

Concentration of data at the very lowest level of measurement has, and will probably continue to be performed on the raw data by bit-oriented algorithms. After the basic measurements have been converted to engineering units, however,

there are data concentration possibilities, based on significance and propagated significance, that affect communications, and the stimulus and response of the system.

GENERAL ADVANTAGES OF DATA ABSTRACTION APPLY

The systems supporting the future I & T applications described in the last section, will be highly distributed. They will contain components from several levels of integration and will also need to be programmed at the highest level possible. The programs which drive the higher-level system functioning should not be bogged down with detailed data analysis. Facilities supporting the propagation of information concerning the validity of measurements and the validity of algorithms concerning those measurements should be basic to the system. Complex programs integrating the functioning of a distributed I & T system will be inherently more maintainable and reusable if kept at highest possible level of data, system and resource abstraction. Greater readability and verifiability of software components, and greater reliability and ease of validation of the system code is then possible. Finally, the design and development of the user interface level applications becomes easier, the higher the level of abstraction that is achieved for the system components and measurements.

ACKNOWLEDGEMENT

I gratefully acknowledge the support given by the Kennedy Space Center/ Engineering Development/ Systems Integration Branch in supplying the computer facilities for the feasibility studies that provided the basis of this work. I also thank my wife, Bronwen Chandler, for her support.

REFERENCES

1. Johnson, C., 1986. "Some Design Constraints Required for the Assembly of Software Components: The Incorporation of Atomic Abstract Types into Generically Structured Abstract Types", Proceedings of the First International Conference On Ada* Programming Language Applications For The NASA Space Station.

2. Karr, M., and Loveman, D. B. III. May 1978. "Incorporation of Units Into Programming Languages", Communications of the ACM, Vol. 21, No. 5.

3. Hilfinger, P. N. 1983. Abstraction Mechanisms and Language Design. Cambridge, Massachusetts: The MIT Press.