Chizeck HJ, Butterworth E, and Bassingthwaighte JB. Automated Unit Balancing in Modeling Interface Systems. IEEE Special Issue (Editors: S. Demir and RWhite), 2009 (In press).

Automated Unit Balancing in Modeling Interface Systems

Howard Jay Chizeck[1,2]*, Erik Butterworth*[1] *and James B. Bassingthwaighte*[1]
Departments of Bioengineering[1] *and Electrical Engineering*[2]*, University of Washington, Seattle WA 98195*

Abstract

The most elementary check on equations used to model physics, chemistry or biology is to determine whether or not the units balance on each equation. When disparate models are combined into more comprehensive or multi-scale systems, conflicts in units commonly occur amongst the units used, such as kg versus g or kPa versus mmHg. Automated unit balance checking followed by unit conversions facilitates the effort of combining models. There are three unit balancing processes: (1) the identification and rejection of model equations containing invalid unit arithmetic; (2) the insertion of scalar multipliers into the equations to convert existing units into fundamental units; and (3) the assignment of units to variables and constants whose units are undeclared, based upon context. Used together these three methods automate the otherwise tedious and error-prone process of manual unit balance checking, thus producing more accurate and readable models. The approach is general and can be incorporated into modeling platforms as an essential first step in model verification. An example is the use in the JSim system for data analysis.

Key Words:

Reproducible research, model verification, automated unit conversion, ontology of unit systems, SI, cgs, MKS, collaborative research and development, scaling conversions, numerical methods, simulation technology, JSim.

Introduction

Computer models of physiology are based on physical laws governing chemical reaction kinetics, magnetic fields, hydrodynamics, ion fields, charge transfer, and other phenomena. Equations that instantiate these models represent the relationships among terms denoting unitary physical properties, such as mass, distance, force, pressure, chemical concentration, and potential difference, along with terms that denote algebraic combinations of unitary properties, such as meters/sec^2 and gram/ml.

The physical properties denoted by the terms of the model equations may be expressed in various units, at various scales: for example, pressure may be expressed as pascals, kilopascals, atmospheres, torr or mmHg, bars and microbars, pounds per square inch, etc. Models often refer to experimental values expressed in units that are convenient to use during the collection of data, and vary in form or in the unit system used.

Most mathematical models published in peer-reviewed journals are not reproducible: they contain the authors' errors of commission and omission, augmented by the errors introduced by editors and typesetters. An exactly reproducible paper is a rarity. Modeling in cardiac electrophysiology has set a high standard of reproducibility in the works of Hodgkin and Huxley (1952)[1], Beeler and Reuter (1977)[2], Winslow, Rice and Jafri (2000)[3], Michailova and McCulloch (2001)[4], all of which can be reproduced, figure by figure, from the papers. The latter two were vetted in advance of publication by reviewers who iterated with authors to assure that the computer code matched the equations appearing in the paper. In all of these studies, the published equations fulfilled unitary balance requirements.

The modeling of biological systems requires validation to assure the reader that the model is a good representation of the real biological or mechanical system, even though it is necessarily a simplification. This means that there is an element of arbitrariness in each of the equations and that there is no easy way to affirm the correctness of the derivations or the assumptions on which they are based. In addition, there is a need to verify the code used to embody the mathematical model. This includes definitions of variables and units. Papers in the mathematics literature are easier to compare, for the development of a theorem can be reviewed exactly.

Given that every biological or physico-chemical model is a hypothesis defining a concept rather than an exact reality, reproducibility is critical to the advancement of the science: A hypothesis disproved is a stepping stone toward an improved hypothesis; that is, toward a better model. The goals of publication in science are manifold, but idealistically the central one is to make the advancements known, exactly, so that they can be challenged and built upon.

To this end we now see the development of model databases and of tools for modeling. What we would expect of databases such as CellML and Biomodels (these are archives of published models using ordinary differential equations in physiology and systems biology) is that the models be reproducible and correct. However many fail; being taken from the published papers, they contain the errors in the papers as well as errors introduced in the process of putting them in the markup language.

The ideas discussed here are not new, having been formally introduced in Buckingham's classic 1914 paper [5] and captured in Wittgenstein's 1922 Tractatus Logico-Philosophicus [6], a work begun in 1914. Both were cognizant of earlier uses of dimensionless variables such as the Reynold's Number for characterizing fluid flow in tubes [7], as reviewed by Sterrett [8]. In 1941 Gagge Burton and Bazaett [9] introduced a system of unitary measurements for thermal exchange in physiological systems. Pappenheimer [10] collated a system for respiratory physiological terminology, including units, in 1950. A 1986 terminology for transport phenomena in physiological systems [11] extended the nomenclature and emphasizing consistent physical and chemical units, allowing for different unitary standards and their inter-conversion. In 1978, Karr and Loveman [12] were the first to directly address the difficult but highly desired capability to incorporate units into programming languages. Their work is an early example of the type of operations described in this paper. Gruber and Olsen [13] proposed an ontology for engineering models which addresses many of the underlying philosophical issues involved in unit balancing. Novak [14] presents elegant methods of automated unit conversion and unit balance checking.

Any Mathematical Modeling Language can be summarized as a collection of variable declarations and mathematical equations using those variables. Variables should be assigned units as they are declared. Variables with no assigned units are declared dimensionless. The tasks of interest in automated unit matching involve:

1. **Equation balance checking**. Equations containing improper unitary arithmetic (e.g. meters/sec + meters/sec^2) should either generate error messages (for manual correction).
2. **Unitary scaling factors**. These should be inserted into calculations as needed (e.g. 6 cm/min = 1 mm/sec).
3. **Unit Inference.** Model variables and constants whose units were not specified should be assigned units based upon context, wherever possible. For example, if A has been assigned the units cm then when the expression (A+B) appears, it implies that A and B have compatible units, and thus B is assigned units cm.

Each of these tasks is described below in greater detail.

An automated unitary correction system should meet the following goals:

1. User specification of variable units should be simple, intuitive and unambiguous.
2. Variants from the basic SI and cgs units such as commonly used units for physiology and systems biology must be predefined in terms of the basic units.
3. Modelers may define new units or abbreviations if desired.
4. Unitarily incorrect equations must be detected.
5. Unitary scaling factors must be automatically inserted into computations.
6. A change to a variable's unit assignment will cause computations to rescale correctly (unitary scaling invariance).
7. In the near term, the system should be accommodate existing models that do not use automated unitary correction technology; that is the unit balance checking should be optional (until modeling standards are adopted generally).

Unit Balance Checking

Unit balance checking is the first of a series of checks for achieving correctness and validity of models. It is the simplest, and is much easier that achieving balance of mass, energy, or charge. It precedes the steps of validation of the model against the observed data, and is a prerequisite for them; there is no point "validating" a partially incorrect model. A checklist for "Standards" can be found at (http://www.physiome.org/standards). Failure to balance means that the equations are consequently erroneous. One source of the error is that units may not have been specified, so that they were inferred from context. The context may be ambiguous, or an inference may depend on units defined by a previous inference. Since the specification of units is the duty of the original authors, there is no good rationale for the failure to define the units for parameters and variables, nor for physical or chemical constants. Another source of error is the use of conflicting unitary systems. Methods to address this are described by Chen et al [15].

In complicated models, balancing the units in equations with many terms is tedious and time-consuming. Constructs such as (meters/sec + meters/sec^2) are inherently erroneous and though they will be detected by a unit balance check, they still must be corrected by the programmer. In standard programming languages like FORTRAN, C, and MATLAB, when the programmer uses centimeters in one equation and meters in another, one usually has to insert numeric unit conversion factors into the model code. It is difficult to find even overt errors since these languages have no provision for unit balance checking. It is also a nuisance to have to convert everything to a common unitary base especially when the experimental observations are best remembered in the format in which they have been acquired. For correctness in equations, each group of terms to be summed in a particular equation must have *identical units at identical scales*, and units to the left of an equal sign must be the same as those on the right.

The incentives to **automate** unit balance checking in computer programs are not just for the perfectionist, but for the practical everyday desire for reproducibility in a quality product. The development of standards for the performance and reproducibility of algorithms for computer modeling and simulation is in its infancy.

What we show here is that unit balance checking can be automated. Although our example implementation is under a specific simulation system, JSim (http://www.physiome.org/jsim/), the program to automate the unit balance checking is general and can be applied to other systems in which units can be specified and checked. While JSim is apparently the first simulation system that did this (in its 1999 release), others are implementing similar improvements to numerical simulation packages. For example, the caretakers of CellML[3] are implementing unit balance checking [16] in an ODE-based simulator Physiome CellML Environment, PCEnv, (http://www.cellml.org/tools/pcenv/. The methodology we describe here exploits that fact that unit balancing follows rigid, logical rules and thus can be automated. Automated unit balance checking not only reduces error in individual equations, but by alerting the model builder to imbalances it also helps to identify any conceptual errors in their formulation.

Unit compatibility and scaling factors

"Unitary scaling invariance" is the property of a mathematical modeling system such that numeric calculations remain correct under changes of unitary scale. Consider a variable V that is specified in volts when a model is first developed, but is changed to millivolts later to conform to usage for a particular application. A unitary scaling invariant modeling system responds by adjusting all calculations using V to be correct under the new scaling. Unitary scaling invariance is also a desirable property for component-based model building where new models are built from a set of simpler modules previously constructed. If variables in the composite model are represented at different scales (e.g. millivolts, microvolts) in the original component models, then a unitary scaling invariant modeling system reconciles them automatically. This capability thus improves prospects for retaining modularity when developing multi-scale models.

Chizeck HJ, Butterworth E, and Bassingthwaighte JB. Automated Unit Balancing in Modeling Interface Systems. IEEE Special Issue (Editors: S. Demir and RWhite), 2009 (In press).

Two units are said to be *compatible* if one can be converted to another via a dimensionless scaling factor. For translating millimeters per minute to centimeters per second the scaling factor is 0.1 cm/mm divided by 60 sec/min or simply 0.1/60 since the components cm/mm are both length measures and sec/min are both time measures, and therefore are compatible. However moles per gram and moles per liter are not. Moles/gram times grams/ml would be compatible with moles/liter.

The *scale factor* is used for converting compatible units such as cm/sec and mm/min. The *dimension vector*, whose length is equal to the number of fundamental units in the model, is used for determining compatibility. Two units are compatible if their dimension vectors are identical.

Inference of Implicit Units

Ideally there should be no need for inferring units. The need to do so arises because, when taking ratios or other mathematical combinations of quantities having well-defined units, it is convenient to derive the units for the resulting products. Automatic assignment carries risk, however. If there are multiple assignments of units to be made in a given equation, it is essential to remove the ambiguity that arises from excessive degrees of freedom.

It is better to assign units to all variables and parameters, including those of computed quantities of interest. The model archiving markup languages CellML [3] and the Systems Biology Markup Language SBML [4] allow, but do not require, unit specification for variables. The practice of designating variables as dimensionless ,when they are really dimensioned quantities, precludes unitary or other balance checking. The choice here is a practical tradeoff between user convenience, conciseness and unitary scale invariance.

Example: The JSim Modeling and Analysis Interface System

We describe here a simulation system that accomplishes the above design goals and tasks. It represents one realization of the unit balancing functions. JSim [17] is a Java-based [18] simulation system for building quantitative numeric models and analyzing them with respect to experimental reference data. JSim was developed primarily for generating model solutions for use in designing experiments and analyzing data in physiological and biochemical studies, but its computational engine is general and equally applicable to solving equations in physics, chemistry, and mechanics. JSim has been under development at the National Simulation Resource (NSR) for Mass Transport and Metabolism since 1999. JSim uses a model specification language, MML (for Mathematical Modeling Language), which supports ordinary and partial differential equations, implicit equations, integrals, summations, discrete events, and allows calls to external procedures. JSim's compiler translates MML into Java code in which the numeric results are calculated. Within the JSim graphical user interface (GUI) users adjust parameter values, initiate model runs, plot data, and perform behavioral analysis, sensitivity analysis, parameter optimization for curve fitting. Alternatively one can use JSim's command line interfaces (jsbatch and jsfim). JSim's capabilities are more advanced than previous NSR software systems SIMCON [19], for simulation control, and XSIM [20] for X-terminal operation. JSim source code, binaries (for Windows, Macintosh and Linux), documentation, and the defining file, *nsrunits,*[21] are available free for non-commercial use at http://physiome.org/.

For purposes of describing JSim's automated unitary correction, the Mathematical Modeling Language (MML) can be summarized as a collection of variable declarations and mathematical equations using those variables. MML allows, but does not require, using physical units in the declaration of the variables. If unitary correction is turned on (this is also optional), then JSim's compiler will perform the following operations while translating MML to executable Java code:

1. Equations containing improper unitary arithmetic will generate error messages (for manual correction).
2. Unitary scaling factors will be inserted into calculations as needed.
3. Model variables and constants whose units were not specified in MML will be assigned units based upon context, wherever possible.

MML unit definitions and variable unit specification

Before an MML variable's unit can be specified, the units themselves must be defined. Units are either fundamental or derived. Fundamental units are defined first. Derived units are defined in terms of previously defined fundamental and derived units. The following MML example fragment defines three fundamental and six derived units:

```
unit meter = fundamental, kilogram = fundamental, sec = fundamental;
unit m = 1 meter, cm = 1/100 meter;
unit g = 1/1000 kilogram,  kg = kilogram;
unit newton = 1 kg*m/sec^2;
unit dyne = 1 g*cm/sec^2;
```

Model writers can define units any way they wish but, for model-to-model compatibility, using JSim's common unit definition file (nsrunit.mod, [21]) is strongly recommended. The file, nsrunit.mod, defines 121 units and 21 decade prefixes (milli, micro, etc.) following "Terminology for mass transport and exchange" [11] and the CellML specification [3]. It defines 7 fundamental units, following the SI (MKS) convention: kilogram, meter, second, ampere, kelvin, mode, candela. The choice to use MKS as fundamental instead of CGS is arbitrary, but of no consequence for model writing since units in both systems are included. Modelers can define new fundamental or derived units in addition to those in nsrunit.mod as needed. For example, many English system units (gallons, miles, etc.) are not defined in nsrunit.mod. It is recommended, in order to make model code most easily understandable, that modelers use the standard terminology in nsrunit.mod wherever possible.

Once units are defined, units for MML variables are specified by combining units into algebraic expressions using the multiplication (*), division (/) and exponentiation (^) operators. For example:

```
import nsrunit;                 // import standard definitions
unit gallon = 3.7854118 liter; // define gallons, needed for this model
math main {                     // start of main calculation section
```

```
    real F = 5 gallon/min;        // F is flow in gallons/minute
    real g = 9.8 m/sec^2;         // g is gravitational acceleration
    ...                           // rest of model omitted
```

Unit compatibility and scaling factors

Each unit in a JSim model is represented internally by a Java data structure consisting of a scale factor and a fundamental dimension vector. These are defined in double precision:

```
double scale;  // scale factor, e.g. 100 for cm when m is fundamental
double[] dim;  // fundamental dimension vector
```

JSim calculates unit scale factors and dimension vectors using the following rules. For fundamental units, the scale factor is set to 1.0 and the dimension vector is set to all zeroes except for 1.0 in the vector element corresponding to the fundamental unit itself. In nsrunit.mod for example, the second fundamental unit is meters, so the dimension vector for meters is [0,1,0,0,0,0,0]. For derived units, the scale factor and dimension vector are calculated using the values of the units from which they were derived. The rules below are applied in order of standard algebraic precedence (parentheses first, then exponentiation, then multiplication & division, with ambiguities resolved from left to right):

RULES:

A) Multiplying by a constant (e.g. min = 60 sec): Multiply the original scale factor by the constant, the dimension vector is unchanged.
B) Multiplying two units (e.g. cm*gram): Multiply the two original scale factors, add the original dimension vectors element by element.
C) Dividing two units (e.g. cm/sec): Divide the two original scale factors, subtract the original dimension vectors element by element.
D) Exponentiation (e.g. cm^3): Raise the original unit's scale factor to the exponent, multiply each original dimension vector element by the exponent.

Consider the processing of unit "grav" (representing gravitational acceleration) in following example:

```
unit kg=fundamental, meter=fundamental, sec=fundamental;
unit cm = 1/100 meter;
unit grav = 980 cm/sec^2;
```

Exponentiation (sec^2) has the highest precedence in the g definition expression. After that, multiplication (980 cm) and division (cm/sec^2) have equal precedence and are processed left to right. Calculations for *grav* proceed as in Table 1:

Table 1. Units in Vector Form

Unit	Scale factor	Dimension vector	Rationale
cm	.01	[0,1,0]	previous definition
sec	1	[0,0,1]	previous definition
sec^2	1	[0,0,2]	exponentiation (rule D)
980 cm	9.8	[0,1,0]	constant multiply (rule A)
980 cm/sec^2	9.8	[0,1,-2]	unit multiply (rule C)
grav	9.8	[0,1,-2]	new definition

Units are considered dimensionless if their dimension vector is uniformly zero. Units are said to be compatible if their dimension vectors are identical (within a machine rounding error of 10^{-7}). For example, accelerations in m/min^2 and cm/sec^2 would both have a dimension vector of [0,1,2,0,0,0,0], although their scale factors (1/3600 and .01) would differ. However, speed (e.g. cm/sec) would have a dimension vector of [0,1,1,0,0,0,0], and thus be incompatible with the accelerations above.

Compatible units are converted by multiplying by the ratio of the scale factors. For example, conversion from m/min^2 to cm/sec^2 is accomplished by multiplying by cm/m (=100 = 1/scale factor) and dividing by sec^2/min^2 (=3600), with the result (cm/sec^2)= (m/min^2)/36.

Basic unitary correction of equations

MML models declare either "unit conversion on" or "unit conversion off". In the former case, the compiler checks for unit compatibility in each algebraic operation, rejecting incompatible ones and inserting appropriate any needed conversion factors into compatible ones. In the latter case, compatibility is not checked and no conversion factors are introduced (i.e., units are only for documentary purposes). The choice of unit conversion declaration is important because correct equation formulation differs in the two cases. For example, if A (in meters) and B (in centimeters) are equated, the correct MML code is as in Table 2:

Table 2. Compare Unit Conversion ON or OFF

with unit conversion on	with unit conversion off
A = B	A = B/100

The remainder of this description will consider only with the case where unit conversion is on. Unit declarations are optional for MML variables and constants. We will first describe processing when units are declared for all variables, and later consider how to deal with missing unit declarations.

JSim's compiler starts by parsing each model equation into a tree based on operator precedence. MML operator precedence (Table 3) is similar to that of many computer languages (C, Java, etc.). Precedence ambiguities are resolved left to right.

Table 3. Operator Precedence

Operator	**Meaning**
()	parenthetical groupings
=	equality
^	exponentiation
* , /	multiplication, division
+ , -	addition, subtraction

For example, consider the following model:

```
unit conversion on;
import nsrunit;
math example1 {
  real A = 2 meter;
  real B = 30 sec;
  real C = 1 min;
  real D cm/sec;
  D = A / ( B + C );
}
```

Following the MML precedence rules, the equation for D is parsed into the tree as in Figure 1:

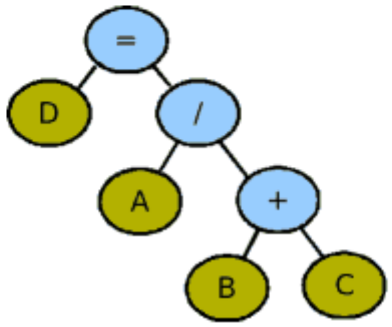

Figure 1. Operator tree.

The tree consists of internal nodes (light blue) which represent operators and leaf nodes (dark olive) which represent the four variables. Internal nodes have two children (left and right) in this example, any positive number in the general case. Internal nodes are examined, depth first, for unit compatibility. Addition and equality nodes require unitary compatibility of their children, but division nodes do not. When compatibility is required, internal nodes are assigned the units of either the leftmost or rightmost child, and an appropriate unitary scaling factor is inserted into the tree. Where compatibility is not required, internal nodes are given a unit appropriate with the operation (here, division). The choice of associativity (leftmost or rightmost child) is arbitrary, but results in equivalent calculations, as by Figure 2:

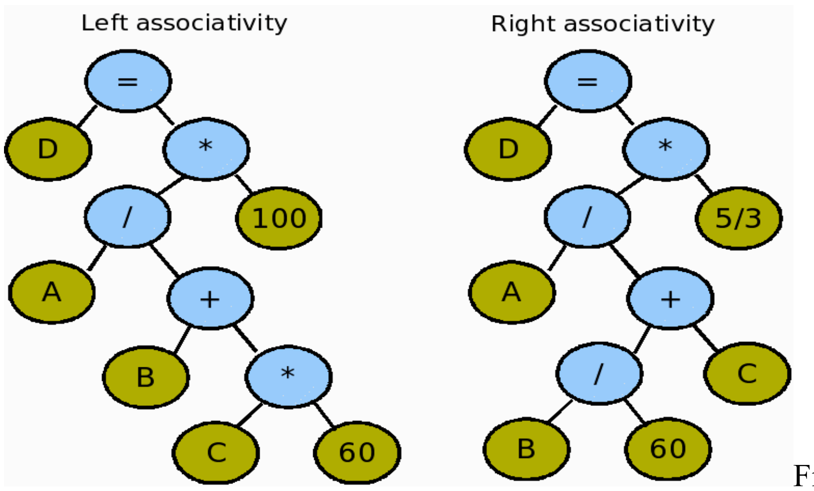

Figure 2. Associativity

The resulting calculations are as follows (Table 4):

Table 4. Equivalent Associativities

Left associativity	**Right associativity**
D=(A/(B+C*60))*100=20/9	D=(A/(B/60+C))*(5/3)=20/9

MML operator and function unitary conversion summary

Table 5 summarizes how the JSim compiler handles unitary conversion for MML's predefined operators and functions.

Table 5. Compiler sequencing of unit conversion.

Operators	**Unit of result**	**Argument requirements**
add(+), subtract(-), equals(=), comparison(<, <=, > & >=), remainder(rem(x,y)), arctangent(atan(x,y))	leftmost child's unit	unitary compatibility
absolute value: abs(x)	same as argument	none
Multiply (*), divide (/)	see unit multiply/divide rules A, B and C above	none
Derivative (:)	same as divide	none
Power (^)	see unit exponentiation rule D above	base is unrestricted, exponent must be dimensionless
square root: sqrt(x)	see unit exponentiation rule D above with exponent=1/2	none
transcendental functions: exp(x), log(x), sin(x), sinh(x), arcsin(x), etc.	dimensionless	argument must be dimensionless

truncation: floor(x), ceil(x). rounding: round(x)	dimensionless	argument must be dimensionless

The dimensionless requirements for truncation and rounding are motivated by unitary scaling invariance. For example, the following model will give a different value for B if A is declared as 0.5 kilogram instead of 500 grams, which should be equivalent:

```
real A = 500 gram;
real B = round(A);
```

The dimensionless argument requirement for transcendental functions is motivated by both unitary scaling invariance and by their Taylor expansions, which are unbalanced if the argument has non-trivial units, e.g.

```
exp(x) = 1 + x + x^2/2 + x^3/6 + ...
```

Radians are defined as dimensionless in nsrunit.mod for modeler convenience. The alternative would be to require MML writers to convert every trigonometric function argument to radians, which we consider verbose and awkward. Steradians are treated similarly. Centigrade and Fahrenheit temperature scale conversions require additive factors that are not handled by the methodology described here. JSim models use the Kelvin scale.

Handling undefined units

Using unit declarations for MML variables and constants is not absolutely required, and missing units will be assigned by the compiler based upon equation context. In the following model, C is automatically assigned units m/sec (to match the right hand side of the equation) and the constant 1 will be assumed to be in seconds (to match B), and listed with values on the Input list, resulting in the calculation of C=10 m/sec:

```
real A = 60 m;
real B = 5 sec;
real C = A/(B+1);
```

In JSim's MML, the above formulation is acceptable shorthand. The completely specified equivalent model below makes clear the writer's intention, using the unit *sec* within a parenthesis along with the 1:

```
real A = 60 m;
real B = 5 sec;
real C m/sec;
C = A / (B + (1 sec));
```

JSim's automated unit assignment algorithm proceeds as follows. A parse tree is generated for each model equation and searched for internal nodes that require unitary compatibility (e.g.

addition, subtraction, equality). If one of the node's children has no unit assigned, it is assigned the unit of the other child. If, after processing all nodes in this way, some nodes are still missing unit assignments, one arbitrarily chosen variable or constant is assigned the dimensionless unit and the entire process is repeated. Eventually all nodes are assigned units. However, it is possible that variable units assigned in one equation may cause other equations to become unitarily unbalanced. If so, the compiler aborts with a diagnostic error message.

Modelers have generally found the JSIM system easy to use. They do not need the technical understanding of the internal calculations described above to balance units properly. The system helps them find conceptual errors in their equations and allows them to inter-mix variables without adding conversion factors. The absence of conversion factors makes their code more readable.

Modelers prefer not having to explicitly assign units to every constant in a model, especially when it is easily understood from the variable name or from the context of the equation. The JSim compiler therefore assigns an appropriate unit, choosing the fundamental unit for it from the context of the equation in which it is used, and using unit conversion. If a user prefers to think of the particular parameter in his experimental units, then he can add the definition of this unit to MML variable declaration and define it relative to units defined in nsrunits.mod. JSim's compiler currently rejects models that incompatibly redefine units in nsrunits.mod, even if the common definitions file is not explicitly imported. This capability will be provided in a future JSim version.

Utility for model reviewing: JSim's unit balance checking is a useful tool for preliminary checking of models that are downloaded from model databases whose formats support unitary assignment. The CellML library is larger but is at an earlier stage of curation. Units are now used in the CellML files. Of the 359 available for download, approximately 60 pass the JSim unit balance checks. However we have not checked the resulting numerical solutions against the original papers.

Many submitted papers contain unitary balance errors that are easily made evident by the reviewer programming them in JSim. Authors can then be guided to fix the problems, which frequently lead to modifying the illustrations. Finding all such errors without using a unit balance checking system is difficult in general, but is a problem readily avoided by using automated checking.

A difficulty is that though a published model may be technically correct, it will not automatically pass the checking for unitary balance if it contains transcendental functions of variables with units, because JSim requires that these be dimensionless. An “error” is detected, and each one of these occurrences must currently be “corrected” by making the argument of the function formally dimensionless. For example, for sin(V) with V in mV, one should write “sin(V/(1 mV)).

The expected usage is to normalize a transcendental argument via a reference value. Sinusoidal functions of time are ordinarily no problem since they are generally written

Chizeck HJ, Butterworth E, and Bassingthwaighte JB. Automated Unit Balancing in Modeling Interface Systems. IEEE Special Issue (Editors: S. Demir and RWhite), 2009 (In press).

```
      sin(2*PI*f*t)
```

where f is a frequency with units of reciprocal time and t is time. The problem comes with such expressions as

```
      exp((V-V0)/10 - 1)
```

where V and V0 are in millivolts, so this requires rewriting

```
      exp((V-V0)/(10 mV)-1)
```

to render the argument dimensionless.

Example Application

In development and analysis of the large cardiovascular/respiratory model VS001 of Neal and Bassingthwaighte, a subset of which is published [22], we have found that automated unitary correction helps to find equation typos such as missing terms or parentheses. VS001 is a closed loop cardiopulmonary model composed of a four-chamber varying-elastance heart, a pericardium, a systemic circulation, a pulmonary circulation, airway mechanics, baroreceptors, gas exchange, blood gas handling, coronary circulation, peripheral chemoreceptors and selectable physiological changes. The VS001 model code contains 846 equations relating 718 terms expressed in 64 different units. This model is available for free download [23].

Consider this example taken from the gas exchange and intracellular buffering part of the VS001 model: the expression in bold should be enclosed in parentheses, but is not, thus evoking an error message identifying an imbalance in units and giving the line number which the error is found:

```
      PBC_pc:t = (Fpc/Vpc)*(PBC_sc-PBC_pc)
        + kp5*CtCO2_ao*(Cheme-PBC_pc)*SHbO2_ao/(1+H_ao/K3bgh)
        + (1-SHbO2_ao)/(1+H_ao/K2bgh) - kp5*PBC_pc*H_ao*(SHbO2_ao/K6bgh
      + (1-SHbO2_ao)/K5bgh);
```

Omitting the parentheses results in an equation that is algebraically seemingly acceptable, but unitary balance fails:

```
   kp5*CtCO2_ao*(Cheme-PBC_pc)*SHbO2_ao/(1+H_ao/K3bgh) has units moles/m^3;
```

Since

```
      (1-SHbO2_ao)/(1+H_ao/K2bgh)
```

is dimensionless, adding the two quantities (as in lines 2 and 3 above) results in a unitary balance error. The error message identifies the first term of the third line as having different units than the other parts of the equation. Automated unitary correction thus pinpoints a problem that would be difficult to find by analysis of the model output. The error message also identifies the nature of the units for the separate parts of the equation, but does not go so far as to recommend where the parentheses should be placed.

JSim unit balance error messages arising from complicated algebraic expressions are sometimes difficult to interpret. This is because unit balance is checked in terms of fundamental units (e.g. kg-m^{-1}-sec^{-2}) while modelers usually think in terms of derived units (e.g. pascals). To address this JSim error messages provide both fundamental and derived unit representations of the offending expressions. However, derived unit representations (unlike fundamental units representations) are not unique, and any single chosen derived unit representation may not correspond to modeler intuition. The current version of JSim (1.6.85) performs minimal simplification of derived unit expressions. Modeler typing mistakes (see above) can result in algebraic expressions with non-intuitive derived unit representation.

Discussion

Unit accounting in modeling is increasingly being recognized as important. For example, JSR 275 [24] is a specification, currently in development, for handling physical units in the Java language that may be incorporated in a future version of the Java language. If so, it will be a valuable addition to the Java language for reliably engineering large systems that deal with quantities in a variety of physical units. In contrast with JSim's current facilities, use of JSR 275 requires significant programming expertise, is rather verbose, provides no facilities for automated unit assignment, and requires complete rewriting of existing computational code.

Such unit checking systems could be further expanded. Incorporating offsets combined with scaling would allow the conversion of temperatures from Centigrade or Fahrenheit to Kelvin. Likewise, pH, pCa and such logarithmic units could be usefully added for modeling in biology.

Units are being incorporated into CellML and SBML. The developers of PCEnv are including this capability in their future developments, as reported in oral presentation at the March 2008 CellML workshop [3]. Unfortunately, standard computational packages have historically lacked such a means of checking, even for space missions. A classic example is the Mars Climate Orbiter mission planning [25] where one team used metric units and the other did not, leading to a miscalculation that put the spacecraft in too low an orbit. It crashed.

The next stage, currently being vigorously pursued by several groups, is to “curate” the models to try to accurately reproduce the results shown in the original publications. The CellML curation team had, as of August 2008, worked on about 365 models from the literature taking them through s series of stages of curation and accumulating 883 model versions in all [3] Of these 616 compiled under JSim and 211 passed unit balance checking. Thus more than half of the 365 models have been rescued from the literature and appear to be correct; this success in difficult work is a testament to the dedication of the curation team The Biomodels curation team (Biomodels 2008) in Cambridge checked over 172 models from the SBML site [4](SBML2008); of these, after a few further improvements, 171 compile though, in many, there is a dependence on inferring units. On the Physiome site (www.physiome.org) there are about 250 models that are archived and downloadable (as are those on the CellML, SBML, Biomodels and JWRmodel sites). Almost all of the JSim versions can be run over the web, with graphics display, parameter adjustment, sensitivity analysis, and even parameter optimization to fit experimental data. Thus they are demonstrably operational. The Physiome models are reproducible and all have unit balance in every equation, since that is a built-in component of the compilation of the model. This is in contrast to the yet-to-be-curated fraction of the models in the CellML and SBML databases. There are also many CellML and SBML models in which physical variables have been labeled dimensionless, so that their meaning is obscured and unit checking is impossible.

A particular virtue of using unit-balanced equations is that when two or more smaller unit-correct models are combined into a larger model, there are no complications matching units between the components, since conversion factors are automatically inserted to bring all into compatibility with the fundamental units. The automated coupling of a set of modules into a composite model requires also that a common ontology be used, that distinct regions, be identified and then that the equations containing variables from more than one module be combined properly. In this situation the component unit definitions directly facilitate the process. A preliminary success in automated combining of two modules has been achieved, and so is feasible.

Multiscale models are becoming ever more abundant as it becomes necessary to link cellular level models to whole organ and whole body models. The cardiac models of Hunter and Noble, the Auckland / Oxford collaboration [26] bring the ion channel cellular depolarization models together with the whole organ finite element cardiac contraction models. The cell models of metabolic reaction sequences in purine nucleosides are brought together into a whole organ model for regional blood flows and capillary-tissue exchanges [27]. These two particular cases are primitive (even if they do now represent tens to hundreds of thousands of ODEs) examples of much larger models to be composed of modules from subcellular domains to whole body controllers.

Building upon the work of others is fundamental to progress in science. Computer modeling and archiving is marvelously efficient for preserving precise descriptions of a current scientific working hypothesis, subjecting it to repeated evaluation tests, and identifying its shortcomings and alleviating them. For quantitative integrative multi-scale models of complex systems, models so preserved are modules to be incorporated into more all-encompassing models which embrace more phenomena and are tested against larger numbers of data sets. Guaranteeing unitary balance and scale invariance in the reference literature and in the models which are the future subsidiary modules is therefore a worthy goal. JSim's unit balance checking is practical tool for this job. Other simulation systems are being modified to include this powerful feature.

References:

[1] Hodgkin AL and Huxley AF. A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117: 500-544, 1952.

[2] Beeler GW Jr and Reuter H. Reconstruction of the action potential of ventricular myocardial fibres. J Physiol (Lond) 268: 177-210, 1977.

Chizeck HJ, Butterworth E, and Bassingthwaighte JB. Automated Unit Balancing in Modeling Interface Systems. IEEE Special Issue (Editors: S. Demir and RWhite), 2009 (In press).

[3] CellML: http://www.cellml.org/workshop/workshop2008/index_html and http://www.cellml.org/ and Cuellar A et al, "An Overview of CellML 1.1, a Biological Model Description Language," *Simulation* 79: 740-747, 2003. DOI: 10.1177/0037549703040939.

[4] SBML, Sytems Biology Markup Language http://sbml.org/Main_Page and Hucka ML, Finney A, Sauro HM, Bolouri H, Doyle JC, and Kitano H. The system biology markup language (SBML) a medium for representation and exchange of biochemical network models. Bioinformatics 19(4): 524-531, 2003.

[5] Buckingham E. On physically similar systems: Illustration of the Use of Dimensional Equations. Physical Review 4: 345-376, 1914.

[6] Wittgenstein L. Tractatus Logico-Philosophicus. Translated from the German by C. K. Ogden, with an introduction by Bertrand Russell. London and New York: Routledge and Kegan Paul Ltd, 1922.

[7] Reynolds O. An experimental investigation of the circumstances which determine whether the motion of water in parallel channels shall be direct or sinuous and of the law of resistance in parallel channels. Proc Roy Soc Lond 35: 84-99, 1883.

[8]. Sterrett SG. Physical Pictures: Engineering Models circa 1914 and in Wittgenstein's Tractatus. http://philsci-archivepittedu/documents/disk0/00/00/06/61/PITT-PHIL-SCI00000661-00/Sterrett-UNC-CH-PPTalk2pdf , 2002.

[9] Gagge AP, Burton AC, and Bazett HC. A practical system of units for the description of heat exchange \ of man with his environment. Science 94: 428-430, 1941.

[10] Pappenheimer JR. Standardization of definitions and symbols in respiratory physiology. Federation Proceedings. 9: 602-605 , 1950

[11]. Bassingthwaighte JB et al, "Terminology for mass transport and exchange. Am. J. Physiol. Heart Circ. Physiol. 250: H539-H545, 1986.

[12] Karr M and Loveman DB. Incorporation of units into programming languages. Communic ACM 21: 385-391, 1978.

[13] Gruber T and Olsen GR. An ontology for engineering mathematics. In: 4th Internat Conf on Priniciples of Knowledge Representation and Reasoning, edited by Doyle J, Torasso P, and Sandewall E. Bonn: Morgan Kauffman, 1994, pp 18pp.

[14] Novak GS. Conversion of Units of Measurement. IEEE Transact on Software Eng 21: 651-661, 1995.

[15] Chen F, Rosu G, and Venkatesan RP. Rule-based analysis of dimensional safety. RTA 2003 LNCS 2706: 197-207, 2003.

Chizeck HJ, Butterworth E, and Bassingthwaighte JB. Automated Unit Balancing in Modeling Interface Systems. IEEE Special Issue (Editors: S. Demir and RWhite), 2009 (In press).

[16] Cooper J and McKeever S. A model-driven approach to the automatic conversion of physical units. Softw Pract Exper 38: 337-359, 2008.

[17] Raymond GM, Butterworth E, and Bassingthwaighte JB. JSim: Free software package for teaching physiological modeling in research. Exper Biol 2003 280.5, p102, 2003., and www.physiome.org

[18] Gosling J and McGilton H, "The Java language environment a white paper," May 1996. http://java.sun.com/docs/white/langenv.

[19] Knopp TJ, Anderson, DU, and Bassingthwaighte JB. "SIMCON--Simulation control to optimize man-machine interaction," *Simulation* 14: 81-86, 1970.

[20] King RB , Butterworth EA, Weissman LJ, and Bassingthwaighte JB., "A graphical user interface for computer simulation. *FASEB J:* 9: A14, 1995.

[21] NSR Units: http://physiome.org/jsim/docs/MML_Units_NSR.html. This file is automatically incorporated into each JSim project file.

[22] Neal ML and Bassingthwaighte JB. Subject-specific model estimation of cardiac output and blood volume during hemorrhage. *Cardiovasc Eng* 7: 97-120, 2007.

[23] www.physiome.org/model/doku.php?id=Integrative_Physiology:Highly-integrated_human_with_interventions:model_detail&s=vs001)

[24] Java units specification: https://jsr-275.dev.java.net/

[25] Mars Orbiter. http://www.spaceref.com/news/viewpr.html?pid=2937

[26] Smith, N. P., Nickerson, D. P., Crampin, E. J. and Hunter, P. J., Multiscale computational modelling of the heart, Acta Numerica, 13, 371-431, 2004

[27] Bassingthwaighte JB, Raymond GR, Ploger JD, Schwartz LM, and Bukowski TR. GENTEX, a general multiscale model for *in vivo* tissue exchanges and intraorgan metabolism. Phil Trans Roy Soc A: Mathematical, Physical and Engineering Sciences 364(1843): 1423-1442, 2006.