HydroUnits: supporting dimensional analysis in hydrologic computing systems using sensor-based standards

Paul Celicourt and Michael Piasecki

ABSTRACT

Unit representation in the Transducer Electronic Data Sheet (TEDS) specified in the IEEE 1451 standards is a binary sequence of 10 octets that encode the physical units as a product of the seven Système International base units, plus radian and steradian, each raised to a rational power in addition to an unsigned integer indicating the structure of the unit. While this representation seems trivial, manual compilation is prone to be erroneous and impractical, especially for complex units used in the hydroclimatology field. Hence, the development of a software application to automatically generate this vector represents a critical step to both reduce work load and automate unit conversion. Existing conversion packages for manipulating units fall short in many ways and also suffer from not integrating with a units controlled vocabulary. We developed HydroUnits (Python-based) to compute the vectorial representation for the Consortium of Universities for the Advancement of Hydrologic Sciences, Inc's Observations Data Model units for use in an IEEE 1451.0-based TEDS generator. In addition, the package has been extended to support dimensional analysis, unit reduction and unit conversion and contains provision to convert time series data between different unit systems.

Key words | dimensional analysis, hydrology, IEEE 1451 standards, Python, unit conversion, unit reduction

Paul Celicourt
Civil Engineering Department,
The City College of New York,
New York,
NY,
USA

Michael Piasecki (corresponding author)
Environmental CrossRoads Initiative,
The City College of New York,
160 Convent Ave.,
New York City,
NY 10031,
USA
E-mail: *michael.piasecki@gmail.com*

INTRODUCTION

In response to the rising complexity and variety of transducers (sensors and actuators) and transducer-to-microprocessors communication protocols, the Institute of Electrical and Electronics Engineers (IEEE) sponsored the development of the IEEE 1451 standard (IEEE 1997, 2007; Song & Lee 2008) to introduce a common communications standard. This standard comprises a family of eight (8) (sub) standards, identified as IEEE 1451.X (as of the date of this writing) that provides the common interface and enabling technology for the connectivity of transducers to microprocessors, control and field networks, and data acquisition and instrumentation systems in a plug-and-play fashion (Song & Lee 2008). The standard puts forth the concept of a Transducer Electronic Data Sheet (TEDS) which plays the role of an identification card attached to smart sensors and actuators enabling self-identification, self-description, self-diagnosis, self-calibration, and plug-and-play functionality. While the TEDS are typically stored and shipped on a non-volatile memory embedded in the sensor assembly, this standard expands on this concept actually allowing the TEDS to be stored in other places (Virtual TEDS) within the user's system. This permits traditional analog sensor users to take advantage of the benefits of TEDS without needing to retrofit their sensors with an embedded nonvolatile memory.

Among the information stored in the TEDS for transducers (TransducerChannel TEDS) is the corresponding unit in which they output their measurements or receive data. Efforts to store physical units associated with transducers have been developed by the IEEE 1451.2 Working Group

doi: 10.2166/hydro.2015.075

(IEEE 1997) for two main purposes: to unambiguously define a sensor measurement and also to identify the scaling associated with a sensor's output. A simple solution would be to enumerate the units for a particular system and transmit only the index of the unit selected by the user for each sensor channel. However, this option does not reveal the unit's structure, it also does not accommodate unforeseen units, and every communicating entity would need a copy of the units list (Hamilton 1996). Moreover, the order of the units in the list would need to be tracked and for verification purposes this would need to be checked at both ends, i.e., at the sources (end nodes) and at the destination (base station). In addition, the inclusion of only the unit index in the TEDS does not allow a manufacturer to build transducers with automatically computed built-in calibration constants. For example, a sensor measuring rainfall in millimeters would require the manual insertion of the scaling factor into the Calibration TEDS. This would be even more tedious for a sensor measuring discharge in cubic feet per minute because such a unit has to be converted into the SI unit system before insertion into the TEDS. In other words, the more complex the unit is, the more onerous is the computation of the aggregated scaling factor. Finally, the number of units that would need to be declared is basically infinite (Hilfinger 1988) resulting in potentially onerous and excessive manual typing together with operator overload to perform dimensional analysis checks and subsequent updates of the units list. An alternative solution consists of using a binary sequence of ten bytes to encode physical units (Hamilton 1996; IEEE 1997, 2007). This method represents a unit as a product of the seven Système International (SI) base units and two SI supplementary units (radian and steradian), namely IEEE 1451 base units (see column 3 in Table 1 for the base set of units), each raised to a rational power for most part with a range of ±4. This structure encodes only the exponents of the IEEE 1451 base units as a dimension vector (e.g., joule (m^2*kg/s^2) is [0,0,2,1, – 2,0,0,0,0]). We subsequently address the raison d'être of the first element named unit interpretation. Its major advantages are that it can accommodate unforeseen units and requires only a few bytes to store units independently of how complex they are. Hamilton (1996) discusses and justifies various considerations behind the separation of plane and spherical angles into two distinct dimensions with units of radian and steradian, respectively. For interoperability purposes, we followed the IEEE 1451 units encoding scheme.

In line with our effort to develop virtual TEDS for legacy sensors to be used in a hydroclimatological measurement system presented in Celicourt & Piasecki (2015) and due to the separation of concerns (e.g., separation of presentation layer, business logic layer, and data access layer) in software engineering (Krueger 2001; Saleh & Gomaa 2005) approach based on the Model-View-Controller architecture (Krasner

Table 1 | Base units layout and indexing for some selected works

	Index						
Quantity	**Unit**	**IEEE 1451**	**Petty**	**Juang and Su**	**Brown**	**Novak: Index**	**Novak: Scale**
Plane angle	Radian	0	7	NA	NA	NA	NA
Spherical angle	Steradian	1	NA	NA	NA	NA	NA
Length	Meter	2	0	0	0	0	1
Mass	Kilogram	3	1	1	1	1	20
Time	Second	4	2	2	2	2	400
Electric current	Ampere	5	3	3	3	3	8,000
Thermodynamic temperature	Kelvin	6	4	4	4	4	80,000
Amount of substance	Mole	7	5	5	5	5	800,000
Luminous intensity	Candela	8	6	6	6	6	8,000,000
Money	Dollar	NA	NA	NA	NA	7	80,000,000
Dimensionless	Unity	NA	NA	7	NA	NA	NA

& Pope 1988; Reenskaug 2003, 2007) we took, we came to realize that it is critical to develop a special package to create the unit representation as specified in the IEEE 1451 standards. The physical units representation described above is used in the TransducerChannel TEDS to specify the engineering unit in which the sensor outputs its measurements amongst other information (that are not the focus of this paper). Our objective was to remove the burden of computing both the compact representation and the scaling factor associated with a unit by designing a package to which the standard units are supplied as is. By automatically handling values and dimensions, this package allows us to free the user from the burden to perform proper scaling for both simple and complex units themselves. Additionally, we release the user from needing to determine the appropriate SI base units' exponents to be used in the compact form in the TransducerChannel TEDS and scaling factor in Calibration TEDS. Our package accepts the CUAHSI's ODM (Horsburgh *et al.* 2008) units controlled vocabularies which contain both SI and imperial units while the IEEE 1451 standards limits itself to SI base units and their multiplicative combination. This means that our package is not limited to a particular system of units despite the fact that the IEEE 1451 requires that unit definitions have to use SI conventions. In addition, it automatically computes the multiplicative factor to enable the conversion from the unit system in which the sensor measures to SI as required by IEEE 1451. Hence, the first objective of HydroUnits is to allow the user of our TEDS Creator application to simply pick the unit which the transducer is measuring or supplying its data in and encode that unit into the IEEE 1451-based vectorial representation that the TEDS Creator can use in subsequent steps to complete the TransducerChannel TEDS creation process.

While one objective of data management systems is to provide the units when annotating the collected data, another is that the units must be correctly manipulated during conversion steps, for example, when storing the data into data stores that require the use of a default unit or just based on a user requested unit conversion when inserting or extracting data from data stores. This is not a trivial task however. For example, the Consortium for the Advancement of Hydrologic Science, Inc. (https://www.cuahsi.org/) developing HydroServer application (Horsburgh *et al.* 2010), uses an extensive controlled vocabulary collection of units to address the need to store point oriented time series data mostly originating from sensor networks. The extensive collection arrives from the use of the imperial and SI unit systems in addition to permitting any units that are deemed appropriate and that have been moderated in the system. While this is a workable approach it features a number of problems.

First, there is a never ending list of requests to allow new unit combinations into the system, for example, another unit describing velocity, even though velocity is already covered multiple times. The simple generic formula of length per time translates into a significant number of representations using both unit systems and then powers of, such as kilo-, milli, micro-, and so on.

Second, in some cases velocity is not really the speed of a traveling particle but could also be the speed with which a surface rises; a good example for this is that rainfall intensity is typically given as millimeter/hour (length/time) which are units of velocity even though this is not a lateral movement covering distance in a certain amount of time. In other words, it would be important to retain context because of those quantities that are 'dimensionally equivalent' while not being 'semantically equivalent' (Kent *et al.* 1996).

Lastly, use of a certain set of units vs. another annotating the same measurement is often subjective because users favor certain units but not others, for example, the use of cubic meter per second for discharge which is a common unit to address river flows, but using million gallons per day when assessing the capacity of water distribution networks or pump performance. Hence, it would be much easier if all incoming data were to be stored using a base set of units which would then be converted on the fly whenever a request for data is incoming demanding a different unit configuration. This leads to the second objective of HydroUnits, which is to serve and support user selected output units regardless of the data acquisition system. Key to a system such as this would be the ability to automatically convert, store, and re-convert any units coming in and going out. To support such a system and hence allow the display of any set of units that the user wants, the IEEE 1451.0 Working Group has added two constants to the Calibration TEDS namely the 'SI units conversion slope' and the 'SI units conversion intercept'. Those constants can either be used as

scaling factors in the conversion process or to determine in what units the conversion process outputs the data. The units definition module of our package (*unitsconversionfactors.py* in Figure 1) models such a system (see Figure 2). We capitalized on this model to extend the package and consequently provide support for observations time series conversion (see the section 'A few examples'). In other words, based on the capability to compute vectorial representation of a given unit in TEDS creation context which is used for dimensional equivalence check, we extended the package to reach a third objective which is to allow practitioners to easily convert time series data for equivalent units. In addition, the units definition module serves as a knowledge base of controlled vocabularies with both original units names, aliases, and abbreviation to check for units consistency.

Note that we do not provide details about terminology use in measurements; to this end the reader is referred to Gehani (1977, 1985), Karr & Loveman (1978), Manner (1986), Gruber & Olsen (1994), Novak (1995), and Allen *et al.* (2004) who present adequate background information on quantity, units, dimensions, etc., and rules for dimensional analysis and units conversion. For a better understanding of some of the terminology we use, however, we summarize the '*placeholder*-unit' phrases we use repeatedly in this manuscript paper in Table 2.

REVIEW OF DIMENSIONAL ANALYSIS APPROACHES

Our initial goal is to incorporate units management in a software application to address inconsistent units error detection and correction and permissible operations as well (Gehani 1977, 1985) based on the IEEE 1451 physical units representation approach. Baldwin (1987), Rogers (1988), and Gonzalez & Peart (1993) have proposed a similar indexed exponent-based vector representation of units to perform dimensional analysis but with a much shorter and different set of base units than the IEEE 1451 base units. Brown (2001), Petty (2001) and Jiang & Su (2006), have recently adopted the same dimension vector approach using respectively a 7-tuple, 7-tuple, and 8-tuple (including radian) unit representation scheme (see Table 1 for base units used and their index in the vectorial form. NA in that table simply means that the corresponding base unit or quantity is not part of the selected base units/quantities for the corresponding work) for automated computation and physical units and dimensions consistency checking. Novak (1995), on the other hand, proposed a much more compact representation model as an integer (namely dimension integer) computed as a linear combination of the elements in the vector representation (dimension vector). A serious shortcoming of this method is that if it allows the use of factors that are multiples of ½ for the exponent, two different units could end up having the same dimension if mathematical operations disregard the meaning of the units. As an example, a force (length^1*time^−2*mass^1) is represented by a dimension vector of [1, −2, 0, 1, 0, 0, 0, 0] (according to the indexing pattern in column 7 of Table 1) which gives a dimension integer of 7,961 (the sum of the product of each value in the dimension vector by its corresponding factor in column 8 of Table 1). Disregarding the meaning of the unit, a unit represented by [−9, −3/2, 0, 1, 0, 0, 0, 0] (length^−9*time^−3/2*mass^1) has the same dimension integer of 7,961 which creates ambiguity (see page 654 in Novak (1995) for more details). Cunis (1992) has proposed a method where each elementary unit of measure is associated with a *prime number* instead of a vector index in order to accommodate a larger set of base units. This method seems to be much more consistent compared to Novak's approach when seen in the context of the above issue; however, units with non-integer exponents cannot be represented with this method, for example, the dimension of the unit of noise spectral density (volts per root hertz). Karr & Loveman (1978) proposed a method where the logarithm of equivalent units is written in a matrix with a specific organization and rule. This method, however, poses the same issue mentioned above in Hilfinger (1988) because it requires an infinite number of units to be defined and, more importantly, logarithmic computation together with large matrix manipulation can potentially generate significant computational overhead.

Other approaches to dimensional analysis and unit conversion include the implementation of a general-purpose object-oriented type system consisting of meta-classes, abelian classes and wildcards to express the most common uses of physical quantities in computing systems (Allen

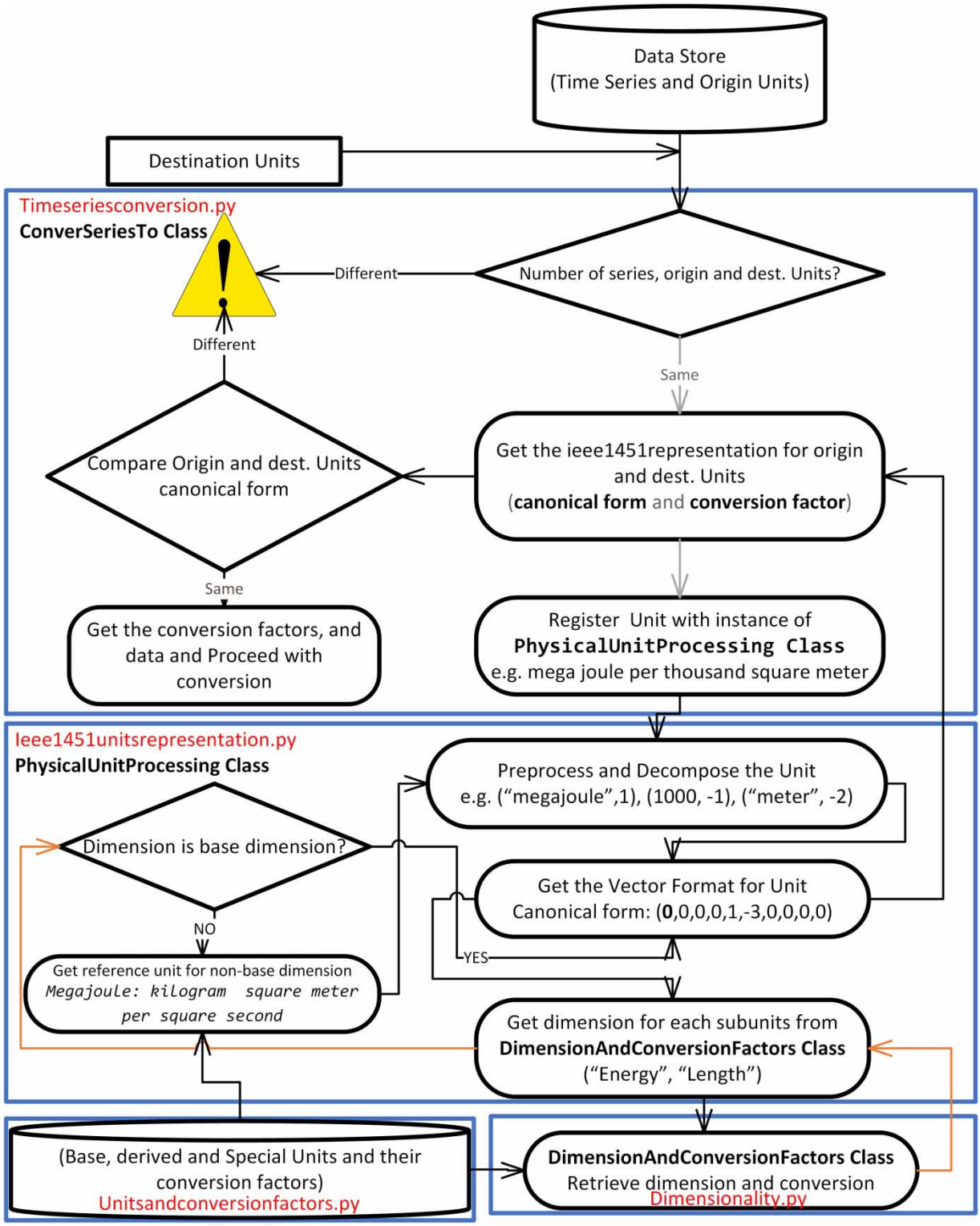

Figure 1 | Diagram depicting the relationship among the HydroUnits packages.

```
"Electric Current" : {"ampere" : {1 : ["A", "amperes", "Amperes"]}
            },

"Thermodynamic Temperature" : {"degree kelvin" : {(1.0,0) : ["degK", "kelvin", "kelvins"]},
                               "degree celsius" : {(1.0,-273.15) : ["degC", "degree Celsius"]},
                               "degree fahrenheit" : {(1.8, -459.67): ["degF", "degree Fahrenheit"]},
                               "degree rankine" : {(1/1.8, 0): ["degR"]}
                               },
```

Figure 2 | Example of data structure to categorize base units according to their dimension.

Table 2 | Definitions of some terminologies used in this paper

Terminologies	Meanings
Base units	Base quantities that are assumed to be mutually independent (e.g., those defined in column 1 of Table 1), such as radian, meter, second, ampere, etc.
IEEE 1451 base units	The nine first units associated with base quantities defined in column 1 of Table 1 and indexed from 0 to 8 in column labeled 'IEEE 1451'
SI base units	The seven units associated with base quantities defined in column 1 of Table 1 and indexed from 2 to 8 in column labeled 'IEEE 1451'
Derived units	Any unit that is expressed algebraically in terms of SI base units including those that have special names (e.g., meter per second, joule, acre)
Given units/Unit-construct	Any unit that is sent as an argument to the HydroUnits package and is subject to be decomposed as a product of base and/or derived units. This terminology mostly refers to units defined in the ODM Units Controlled Vocabularies which consist of base units (e.g., degree kelvin), derived units (e.g., square meter), and more complex ones such as megajoules per square meter per day
Reference unit	A base or derived unit when expressed as a product of the selected base units results in a scaling factor equal to 1 (e.g., newton, joule)

et al. 2004), integration of object-oriented dimensional analysis at the language level (Damevski 2009), extension of strongly typed language with dimension implemented as a type (Kennedy 1994), non-dimensionalization based on the Buckingham Pi theorem (Khanin 2001), and data-abstraction facilities of C++ (Cmelik & Gehani 1988).

The approach to unit representation in smart and legacy sensors specified in the IEEE 1451 standard, while based on the dimension vector representation method, is more streamlined, consistent, and comprehensive to perform dimensional analysis. This approach relies on the concept of 'unit interpretation' as a key instrument to reenforce more accurate dimensional analysis and is used to capture or express whether the intended unit is the given one or the quotient of the given unit and itself or the logarithm of that unit, or the logarithm of the quotient of the given unit and itself, etc. (see Table 3). For example, the strain quantity in meter/meter is represented by [**1**, 0, 0, 1, 0, 0, 0, 0, 0, 0] where the first element in the list

Table 3 | Unit interpretation value and their meaning adapted from IEEE (2007)

Value	Meaning
0	Unit is described by the product of SI base units, plus radians and steradians, raised to the powers recorded. Units for some quantities, such as the number of people through a turnstile, cannot be represented using these units
1	Unit is U/U, where U is described by the product of SI base units, plus radians and steradians, raised to the powers recorded
2	Unit is log10 (U), where U is described by the product of SI base units, plus radians and steradians, raised to the powers recorded
3	Unit is log10 (U/U), where U is described by the product of SI base units, plus radians and steradians, raised to the powers recorded
4	The associated quantity is digital data (for example, a bit vector) and has no unit. The 'digital data' type applies to data that do not represent a quantity, such as the current positions of a bank of switches
5	The associated physical quantity is represented by values on an arbitrary scale (for example, hardness)

indicates that the vector is a quotient of the unit represented by the nine other elements, each one representing an element of base dimension. However, radian (m/m) is represented by [**0**, 1, 0, 0, 0, 0, 0, 0, 0, 0]. This method demonstrates that an important piece of information is lost by saying that a sensor measuring spherical angle in steradians (m^2/m^2) has a unit representation equal to [0,0,0,0,0,0,0,0] as it would have been represented by the aforementioned references except by Petty (2001) because the radian unit is included in this work. With this concept, the IEEE 1451 Working Groups made a step further in allowing the representation of dimensioned, dimensionless, and even digital quantities in a more comprehensive fashion. They also established a way to differentiate quantities/units that would otherwise be considered dimensionless. In IEEE 1451, the term 'dimensionless quantity' is reserved for quantities with units like count (population, trees, etc.) or output (cars, devices, etc.) of a manufacturing company because the thing they quantify is not a recognized dimension. They are then represented in the vectorial form as [0,0,0,0,0,0,0,0,0,0] without the dimensionless ratio flag. This method also expresses both the need for more dimensions definition and also captures the underlying structure of units.

The work cited above demonstrates that significant efforts have been invested into the dimensional analysis problem. In practice, dimensional analysis has also been instrumental for modeling real-world case studies. For instance, dimensional analysis, commonly deployed in the field of hydraulics (Goldberg 2000), was carried out to determine the mathematical formula of apparent shear stress in prismatic compound open channels using artificial neural networks techniques (Huai *et al.* 2013) and for predicting scour depth downstream of culvert outlets with artificial neural network models (Liriano & Day 2001). In addition, it has been used to determine the bed concentration of suspended sediment from flume experiments (Minns 2000) and to identify dimensionless terms associated with underlying relationships between output and input variables in two-phase flow modeling (Morshed & Powers 2000). Furthermore, it has been used as an additional source of information to check the validity and usefulness of relations created on the basis of data in knowledge-discovery software system (Babovic 2009).

BRIEF ANALYSIS OF EXISTING PYTHON PACKAGES TO WORK WITH UNITS

Our choice of encoding environment (Python) does not natively provide any support for our task. However, more than a dozen Python modules (PSF 2014) (*DimPy, Magnitude, NumericalUnits, Python-quantities, Scalar, Scientific.Physics.PhysicalQuantities, SciMath, sympy.physics.units, udunitspy, astropy.units, Buckingham, Units, Pint, natu,* etc.) have been developed to deal with units attached to quantities. A thorough inspection of all the packages mentioned above, in particular of the most prominent ones including Pint (https://github.com/hgrecco/pint), Numerical Units (https://pypi.python.org/pypi/numericalunits), Units (https://pypi.python.org/pypi/units/), astropy.units (Robitaille *et al.* 2013), and BuckinghamPY (https://github.com/mdipierro/buckingham), demonstrates that these packages differ by a wide variety of factors. Those factors include the implementation approaches taken by the authors, the extra functionalities they offer besides unit conversion and dimensional analysis, support for user-defined units, consistency check, sets of supported units as well as the units names and relationships, among others. Also, in most of the cases, the units naming convention is not standard based. For instance, the application of mathematical operations to unit names and mixing of unit name with unit abbreviation is found in Pint which is unacceptable according to the National Institute of Standards and Technology (NIST) (Thompson & Taylor 2008). In addition, many of the packages including units do not have a single reference unit for a specific dimension. Instead, a hierarchical unit definition scheme is used to relate commensurate units (e.g., minute = 60 seconds, hour = 60 minutes, etc.) which has the tendency to generate computational overhead. Some packages such as the Numerical Units package, Astropy. Units features redundancies in supported unit definitions where prefixes are attached to submultiples and multiples which is considered poor practice (Dreiheller *et al.* 1986). Finally, some packages do not provide support for conversion between units with non-multiplicative factors

such as temperature, and also not for units involving amount of substance such as pressure (e.g., millimeter of water). In summary, none of the packages we reviewed incorporated controlled vocabularies or made use of them, nor did any of them natively support dimensional analysis and conversion of units with the naming convention adopted by the NIST (Thompson & Taylor 2008). Hence, we concluded that they are not suitable to manipulate the set of CUAHSI units which include many complex units and, more importantly, achieve the IEEE 1451-based vectorial representation of the units. In addition, there are a number of semantic challenges that arrive from how unit names are spelled out and what abbreviations are used, such as in '*hectometer*' versus '*hecto meter*' and also from the span of modifiers like square, cubic, squared, and cubed which may require the help of the corresponding abbreviation in unit like *millimoles per cubic meter squared* ((mmol/m^3)^2) vs *meter squared per second squared* (m^2/s^2 and not (m^2/s)^2) to capture the structure of the unit. Additionally, most of these packages do what we may call 'meshed dimensional conversion' by analogy to a meshed sensor network where any node (in our case unit) can communicate with any other node in its range (equivalent unit in our case) while we were seeking to perform a 'starred dimensional conversion' (a reference unit is like a coordinator or a router) to satisfy the SI-based IEEE 1451 vectorial representation requirement. Therefore, they do not target a specific base unit to decompose a given unit into in order to create its dimension vector representation and, more importantly, they do not implement a unique reference unit when unit conversion between commensurate units is necessary. Moreover, the adoption of the scale-factor-based hierarchical unit definition scheme used rendered these Python packages unsuitable for our task. There are many other aspects of the comparison that are not covered here; the reader is referred to Hillar (2013) to further explore a more fine-grained comparison between three (Numerical Analysis, Units, and Pint) of the reviewed python modules. Lastly, we discovered various similarities (hierarchical unit definition scheme, redundancies in units definition, no support for units with additive scale factors, etc.) between the approaches adopted in the mentioned Python packages and at least one of the works reviewed in the section 'Review of dimension analysis approaches'.

HYDROUNITS

Key design principles

A number of key design principles are adopted to develop the python module to meet the aforementioned objectives.

(a) **Complete independence from Python's non-standard package**: our application is built solely using Python built-in packages to make installation and utilization as easy and convenient as possible. This design principle allows the application to be lightweight in terms of memory usage as Python is a 'batteries included' language (Oliphant 2007) and also because it takes advantage of the Python 'dynamic typing' (Tratt 2009) capability which permits low development cost and in addition does not require data or object type definition in the context of this work. This also facilitates the installation and execution of our application on resource-constrained devices provided that Python can be installed and run on such devices. Moreover, the application as developed in Python is platform agnostic which means that this application will be able to run on platforms such as Windows, Mac OS, Linux, Android, etc., with no or little alteration.

(b) **Standalone units and conversion factors definition**: Our package has been developed with software reusability, pluggability, parallel implementation of components and modularity in mind. Hence, the Consortium of Universities for the Advancement of Hydrologic Sciences, Inc.'s Observations Data Model (CUAHSI ODM) Units, on the one hand, and conversion factors between simple and derived SI and non-SI units to the base units, on the other hand, are defined in separate files. Keeping these files separated is instrumental to being able to automatically interrogate the CVs we are using for integration of the latest version.

(c) **Python dictionary types**: This Python data type consists of comma-separated key-value pairs within a pair of curly brackets. It allows the capture of more structured information than a simple list of elements and its nesting feature allows building up and accessing complex information structures directly. In the context of this work, this structure is used as a knowledge base to achieve

the 'starred dimensional conversion' task. The strength of this data structure is that it can accommodate and store an unlimited number of Python data structures and types including itself. The representation of the base and derived units and their conversion factors, shown in Figures 2 and 3, follows the hierarchical and categorical pattern below:

Units categories = {'Dimension': {'equivalent units': {'conversion factor': 'alias/abbreviated name of equivalent units'} } }.

Because we programmatically determine the vectorial representation for base units, derived units and any given unit, we chose to group equivalents units under their respective dimension and quantity. This approach offers convenience for the tasks at hand.

Base and derived units definition and conversion factors

We adopt the approach of House (1983) and Novak (1995) by grouping dimensionally equivalent units under a single dimension with a reference unit which is either the base unit defined in IEEE 1451 or a unit derived from these base units. The conversion factor for those reference units is 1.0 or (1, 0). We capitalize on the notion of 'immutable type' (e.g., int, float, tuple, etc.) in Python to define the conversion factor for each co-unit/commensurate unit in the dimension-based group as the dictionary data type (mentioned above) requires that the key for each value is immutable. We also take advantage of the 'dynamically type checking' capability of Python to handle the difference between conversion factors that have no additive factor and those that have one. Hence, we use '*tuple data*' type for those that have additive factor like thermodynamic temperature units (see Figure 2) and '*integer*' or '*float data*' type for those with only a multiplicative scale factor. We also adopted Dreiheller *et al.*'s (1986) recommendation to keep base and derived units and their multiples and sub-multiples as two separate but interconnected entities using the dictionary type to define these scale factors. For instance, we define milli, kilo, hecto, etc., and meter, and our package understands and computes scaling factor for kilometer and even for units that may be written as kilo meter. In addition, there can be numbers (written as a text string) that may appear in some of the units (e.g., kilograms per *thousand* square meter) for which we will use the same data type to define their value. However, we do not manually provide nor define any base unit or derived unit in terms of its compact representation as done in Petty (2001) and Brown (2001) which is prone to errors. Instead, we programmatically compute the IEEE 1451 representation for the derived units in order to avoid erroneous values (see dashed box in Figure 3). Hence, we simply define the name of the reference unit of the group as a function of the base units that appear in its algebraic expression. As for the vector representation of the base units, this is simply done by replacing the corresponding index for the base unit by its exponent in a zeroed 9-tuple, then the unit interpretation is added to make up the 10-tuple. Here HydroUnits offers some versatility in the sense that it can easily be adjusted to support 7-tuple and 8-tuple based vector representation. HydroUnits cannot guarantee that the unit-construct supplied to the package for a specific purpose (unit conversion, dimension computation, etc.) has a physical meaning. We cannot restrict this as it would compromise the ability of

```
"Force" : {"newton" : {1 : ["N", "newtons"]},
           "dyne" : {1e-5 : ["dyn"]},
           "pound force" : {4.448222 : ["lbf"]},
           "ounce force" : {2.780139 : ["ozf"]},
           "kilogram force" : {9.80665 :["kgf"]},
           "kilogram meter per square second" : {}
           },
```

Figure 3 | Definition of the reference derived units as a function of base units with non-zero exponent.

HydroUnits to handle unforeseen units. However, we guarantee that each sub-unit of the unit-construct is defined in the knowledge base and also allow the user to specify his/her own units according to the pattern shown in Figures 2 and 3.

Building blocks of the module

The HydroUnits package contains three modules, namely, *dimensionality.py*, *ieee1451unitsrepresentation.py* and *timeseriesconversion.py* defined in Figure 1. Each module serves a specific but different purpose for which we need to define different data types and formats. For example, the *dimensionality.py* module is used to determine the dimension for each element of base or derived unit that appears in a given unit. Hence, it works only with string data type and with units defined in the 'units' knowledge base (*unitsconversionfactors.py*). It is used as a support for the *ieee1451unitrepresentation.py* module which itself uses any unit similar to those specified in the CUAHSI ODM.

The *ieee1451unitrepresentation.py* module is the main engine of this package and is responsible for carrying out a number of processing tasks including resolution of semantic heterogeneities and replacement of numbers declared as string by their corresponding values for the given unit. It also decomposes the latter into separate sub-units defined in the knowledge base file *unitsconversionfactor.py* to be sent to the *dimensionality.py* module in order to obtain the dimension for each sub-unit. This module contains a method that slices the unit-construct into sub-units that can be tested against the pre-defined CVs in the 'units' knowledge base. This allows us to remove subunits such as 'degree Celsius' and 'international foot' from a hypothetical unit such as 'international foot degree Celsius' (see Figure 4). In addition, it automatically computes the 'unit interpretation' value mentioned above, the exponent for each sub-unit, and the IEEE 1451 vectorial representation as well as the overall conversion factor for the given unit. Figure 1 also provides an insight into the overall process of time series conversion which we address in the section 'A few examples'.

When time series conversion is needed, the entry point is the *timeseriesconversion.py* module that communicates directly with the *ieee1451unitrepresentation.py* which, in turn, constructs the IEEE 1451 unit representation for both the origin and destination units. The *timeseriesconversion.py* will then request the vectorial representation for the origin and destination units and perform a (dimension) check for equality on both vectorial representations before carrying on with the conversion process. As mentioned earlier in this paper, the *unitsconversionfactors.py* module constitutes a unit knowledge base CV that groups base and derived units according to their dimension. It also maps each of those units with their scale factors to enable inter-units conversion.

A few examples

Being able to compute the dimension vector as specified in IEEE 1451 for our TEDS generator (Celicourt & Piasecki 2015), we have extended the package to perform dimensional analysis and unit reduction, and consequently, time series units conversion. Our package supports the following main operations.

Determine the 'basic dimension and conversion factor' to the predefined reference unit for a given base or derived unit (Figure 5)

Determine the 'compact representation and conversion factor of a base unit' (Figure 6)

Determine the 'compact representation' of a given unit as a function of the defined base units. 'Unit Reduction or Simplification' is internally performed (Figure 7)

```
>>> from hydrounits.ieee1451unitrepresentation import PhysicalUnitProcessing
>>> un = PhysicalUnitProcessing()
>>> un.registerUnit("international foot degree celsius inch of mercury")
>>> un.dimension_map
{'degree kelvin': 1, 'Kilogram': 1, 'second': -2, 'steradian': 0, 'radian': 0,
'lumen': 0, 'meter': -1, 'ampere': 0, 'mole': 0}
```

Figure 4 | Demonstration of the unit-construct slicing method.

```
>>> from hydrounits.dimensionality import DimensionsAndConversionFactors
>>> dim = DimensionsAndConversionFactors("pound force")
>>> dim.base_dimension
'Force'
>>> dim.conversion_factor
4.448222
```

Figure 5 | An example of base dimension and conversion vector retrieval for a derived unit.

```
>>> from hydrounits.ieee1451unitrepresentation import PhysicalUnitProcessing
>>> unit= PhysicalUnitProcessing()
>>> unit.registerUnit("degree celsius")
>>> unit.canonical_form
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
>>> unit.dimension_map
{'degree kelvin': 1, 'Kilogram': 0, 'second': 0, 'steradian': 0, 'radian': 0,
'lumen': 0, 'meter': 0, 'ampere': 0, 'mole': 0}
>>> unit.net_conversion_factor
(1.0, -273.15)
```

Figure 6 | An example of base unit vector representation and conversion factor.

```
>>> from hydrounits.ieee1451unitrepresentation import PhysicalUnitProcessing
>>> un = PhysicalUnitProcessing()
>>> un.registerUnit("pound force per square feet")
>>> un.canonical_form
[0, 0, 0, -1, 1, -2, 0, 0, 0, 0]
>>> un.dimension_map
{'degree kelvin': 0, 'Kilogram': 1, 'second': -2, 'steradian': 0, 'radian': 0,
'lumen': 0, 'meter': -1, 'ampere': 0, 'mole': 0}
>>> un.net_conversion_factor
47.880074617221055
```

Figure 7 | An example of IEEE 1451 representation for a given unit.

Time series conversion with support for units with additive factors (Figure 8)

In this example, we use Pandas (http://pandas.pydata.org/), a versatile Python package for data structures and data analysis, to read a data file produced by a Campbell Scientific, Inc.'s CR1000 datalogger. We select two of the time series (Temperature and Solar Radiation) in the .dat data file and the timestamps (see arguments usecols = (0,5,6) of the read_csv method) and create two vectors that are passed as arguments to the *ConvertSeriesTo* Class in our package along with the origin and destination units of the time series. Figure 8 shows the results of the conversion and also depicts how one can keep track of both the origin and destination units after the conversion is performed. The new series of time series or DataFrame in Pandas jargon can be reconstituted into a data file using a suitable Python package such as Numpy, xlrd, the Python built-in function *open()*, etc. The selection of the Python package to read data files is left to the discretion of the user.

Support for units involving monthly values or monthly rates (Figure 9)

HydroUnits also provides supports for time series conversion involving monthly values and monthly rate values taking into account leap years. This has been particularly useful in more

```
Python 2.7.9 (default, Dec 10 2014, 12:28:03) [MSC v.1500 64 bit (AMD64)] on w
in32
Type "copyright", "credits" or "license()" for more information.
>>> import pandas as pd
>>> from hydrounits.timeseriesconversion import ConvertSeriesTo
>>> data = pd.read_csv("c:/campbellsci/loggernet/cr1000_beloc.dat",\
 sep= ",", header=1, skiprows=2,  nrows=10, usecols=(0,5,6))
>>> timeseries = [data.get_values()[:,i] for i in range(1, len(data.columns),
1)]
>>> print "the given time series = {}".format(timeseries)
the given time series = [array([26.62, 26.6, 26.56, 26.55, 26.52, 26.5, 26.22,
25.86, 25.83, 25.84], dtype=object), array([0.012, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0], dtype=object)]
>>> new_data = ConvertSeriesTo(timeseries, ["degree celsius","watt per square
meter"],\
 ["degree fahrenheit", "kilowatt per square feet"])
>>> print "the converted time series = {}".format(new_data.new_time_series)
the converted time series = [[79.916, 79.88000000000005, 79.80799999999994, 79
.79000000000002, 79.73599999999993, 79.69999999999999, 79.19599999999997, 78.5
4799999999994, 78.49399999999997, 78.51199999999989], [1.11484086912432e-06, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
>>> print "the origin unit(s) = {}".format(new_data.origin_unit)
the origin unit(s) = ['degree celsius', 'watt per square meter']
>>> print "the destination unit(s) = {}".format(new_data.destin_unit)
the destination unit(s) = ['degree fahrenheit', 'kilowatt per square feet']
```

Figure 8 | An example of HydroUnits time series conversion results.

accurately converting precipitation rate data (for example) from the National Center for Environmental Prediction (NCEP) Reanalysis project available at: http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.html. Here we simply illustrate this capability in Figure 9.

Support for unforeseen units (Figure 4)

Here we present an example of results of the unit-construct slicing algorithm mentioned before. To show the robustness of the slicing method, we use a hypothetical unit-construct disregarding its physical meaning and we supply it to HydroUnits. The unit reduction expression shown in Figure 4 demonstrates the ability of the overall package to handle unexpected units. The unit reduction result also demonstrates that HydroUnits is able to subtract a sub-unit with more than one string piece like 'international foot' and 'inch of mercury'.

Support for logarithmic quantities (Figure 10)

Providing support for handling logarithmic quantities, more specifically the 'pH' quantity (measure of acidity), in

```
>>> from hydrounits.timeseriesconversion import ConvertSeriesTo
>>> new_series = ConvertSeriesTo([[1,2,3,4,5]], ["millimeter per second"],\
 ["meter per month"], year = 2016, month = 2)
>>> print new_series.new_time_series
[[2505.6, 5011.2, 7516.8, 10022.4, 12528.0]]
>>> new_series = ConvertSeriesTo([[1,2,3,4,5]], ["millimeter per second"],\
 ["meter per month"], year = 2015, month = 2)
>>> print new_series.new_time_series
[[2419.2, 4838.4, 7257.599999999999, 9676.8, 12096.0]]
```

Figure 9 | Illustration of time series conversion involving monthly rate values with leap years considered.

```
>>> from hydrounits.ieee1451unitrepresentation import PhysicalUnitProcessing
>>> un = PhysicalUnitProcessing()
>>> un.registerUnit("ph unit")
>>> un.canonical_form
[3, 0, 0, -3, 0, 0, 0, 0, 1, 0]
>>> un.dimension_map
{'degree kelvin': 0, 'Kilogram': 0, 'second': 0, 'steradian': 0, 'radian': 0,
'lumen': 0, 'meter': -3, 'ampere': 0, 'mole': 1}
>>> un.registerUnit("decibel")
>>> un.canonical_form
[3, 0, 0, 2, 1, -3, 0, 0, 0, 0]
>>> un.dimension_map
{'degree kelvin': 0, 'Kilogram': 1, 'second': -3, 'steradian': 0, 'radian': 0,
'lumen': 0, 'meter': 2, 'ampere': 0, 'mole': 0}
```

Figure 10 | Illustration of logarithmic quantities handling in HydroUnits.

HydroUnits requires a reconciliation of the IEEE 1451 standards and the physical sciences treatment of such quantities. In Table 3, for example, IEEE 1451 allows a dimensional representation for units expressed as log(U) where U can be any given unit. However, such unit-construct has raised intense debate for several decades.

It also demonstrates that, because of the property of the logarithmic function, dimensions are carried additively for a quantity like log (20 meter) which is a misconception (Matta *et al.* 2011). However, the accepted procedure to handle logarithmic quantities is to make the quantities themselves dimensionless (Boggs 1958; Molyneux 1991; Mills 1995; Matta *et al.* 2011). Therefore, to represent the 'pH' quantity which is considered to be outside the scope of the IEEE 1451 standard (Hamilton 1996), we instead adopted the following definition (Boggs 1958; Mills 1995; Matta *et al.* 2011):

$$pH = -\log_{10}\left(\frac{[H^+]}{mol \cdot m^{-3}}\right)$$

Discussions of inaccuracies in the results presented

The reader may notice that HydroUnits exhibits some inaccuracies in the results presented above. The behavior of these errors in floating-point arithmetic is tolerated and it is also not recommended to round or truncate floating-point numbers to fix perceived accuracy problems (Beazley & Jones 2013). In Python, even trivial or simple mathematical calculations generate some level of errors. This is due to the fact that floating-point numbers cannot accurately represent all base-10 decimals. In addition, a floating-point number may have a finite decimal representation, but its binary representation in computer hardware is an infinite repeating representation (Goldberg 1991). All computer hardware and computer programming languages suffer from such limitations. Another option beside rounding or truncating is to treat the numbers as decimal using the *decimal* Python standard module, but this would sacrifice performance at some levels. In addition, rounding or truncating of numbers may be involved. In this regard, we believe that the application domain (e.g., finance vs. engineering or science) matters the most. In the scientific and engineering domains, it is common and encouraged to use the normal floating-point type. Therefore, as mentioned before, HydroUnits makes use of the Python's float data type.

SUMMARY

In this paper, we addressed a number of unit conversion challenges such as: what is the resulting conversion factor for units involving both a subunit with only a multiplicative scale factor and a subunit with both a multiplicative scale factor and an additive constant? For example, how do we convert from a unit such as 'meter per degree Celsius' to 'feet per kelvin'? Also, while it would make more sense to consider a 'temperature delta' instead of a 'temperature point' when the thermodynamic temperature dimension is involved in multiplicative context, do we know enough about how the measurement quantified temperature to

make this choice? These questions illustrate that dimensional analysis is not sufficient to perform a complete and sensible unit conversion. Another example to illustrate this is: arithmetically adding two temperature values yields the sum of these two values. However, in the context of 'heat transfer' when mixing two substances with different temperature, because temperature is an intensive quantity, the temperature of the resulting mixture is not or cannot be the sum of the initial temperatures. Therefore, when dealing with intensive quantities, additional information about the substances to be mixed is paramount to correctly perform the unit conversion.

We also realized that dimensional analysis does neither adequately model the semantics of measurement data nor does it provide adequate treatment of dimensionless quantities and distinguish between such concepts as circular angles and rotational angles, or temperature points and temperature intervals. For instance, quantities such as 'relative humidity' and 'volumetric water content', expressed in 'percent', are treated as dimensionless quantities, but in the context of the IEEE 1451 standard the unit (percent) has to be replaced by its equivalents (respectively 'Pascal per Pascal' and 'cubic meter per cubic meter') to express the underlying structure of the unit and also perform proper dimensional representations. Moreover, using the unit 'percent' for these quantities in computing systems means that they are dimensionally equivalent and consequently one could be converted to the other. Hence, adopting the IEEE 1451 unit representation prevents the mistake of performing such illegitimate automatic unit conversion. One could argue that this resembles an underdeveloped standard in the sense that it does not fully capture the intricate details of the underlying unit (e.g., Pascal water vapor pressure per Pascal saturation vapor pressure). However, the IEEE 1451 standard does not attempt to consider such nuances in the underlying unit which is an unmanageable task. In addition, addressing such level of details would generate some redundancies as only the unit component appearing in the dimensionless ratio is important for the vectorial representation. We also identified some areas for further studies, for example, that new dimensions or strategies are needed to represent water quality and microbiology measurements (water color, turbidity, bacteria population, etc.).

The developed tool (HydroUnits) establishes the basis and direction for more accurate dimensional analysis for data collected in the geosciences. While it is developed to serve in a hydroclimatological context, we believe that it can serve for other fields of study as hydroclimatology is a crossroads to many other fields, at least as far as the units are concerned. We have shown examples that illustrate how this tool is capable of performing dimensional analysis, unit reduction and ultimately conversion between unit systems. This tool, developed in Python language, can serve in both online (with a Python Web Framework such as Django) and offline (in Graphical User Interface and Command line scripts) water data analysis systems. For example, an online water data system analysis can harvest water data from the ODM together with the ODM units CVs associated with the data and perform time series data conversion between different units system. To this end, we have developed a number of smaller packages, namely, one (*unitsconversionfactors.py*) storing base and derived units and some special units (e.g., hectare) along with scaling factors as Python dictionary data types; then a package (*dimensionality.py*) that computes support information necessary for retrieval of the dimension and scaling factor associated with the base and derived units appearing in a given unit-construct such as those from the CUAHSI ODM. Based on these two packages, a user can easily determine the IEEE 1451 representation of a unit. Hence, we developed another specialized package (*ieee1451unitrepresentation.py*) that preprocesses the given unit-construct for consistency checking and further manipulation. Subsequently, this package decomposes the given unit-construct as list of subunit and exponent pair which will be piped individually to our *dimensionality.py* package which itself returns the dimension and scaling factor for the subunit. If the decomposed unit-construct contains a derived or special unit, this is again decomposed and each of its subunits sent to the *dimensionality.py* package computing dimension and scaling factor. In addition, the user may want to visualize data or compute data product with a mix of data time series in different unit systems, hence, we have developed a package named *timeseriesconversion.py* that permits the conversion of such time series into the desired unit.

HydroUnits differentiates itself to existing tools by a number of factors including the implementation approach adopted, the adoption of standard-based (NIST) units naming conventions and, more importantly, the emphasis on units controlled vocabularies which is a critical aspect of units treatment. Additionally, HydroUnits supports unit conversion for quantities with additive scaling factor, and natively supports time series conversion and takes leap years into consideration for units consisting of the time dimension (e.g., month, minute). Due to its overall implementation approach, HydroUnits exhibits a high level of versatility that no other tool we are aware of has achieved. This versatility is illustrated by the following examples:

(a) The dictionary data type used to store the base and derived units and their conversion factor can easily be serialized into the JavaScript Object Notation format making the unitsandconversationfactors.py file contents to be stored in a non-relational database such as MongoDB and the other components of HydroUnits can, with minimal efforts, be paired with a tool like the Pymongo driver (to communicate with MongoDB from Python) to help with dimension and scaling factors retrieval. This would be an important aspect of a (sensor) data management system allowing users to retrieve sensor data from a database (CUAHSI ODM) in their preferred unit system. This makes HydroUnits unique. In addition, for the reasons cited in 'Python dictionary types' in the section 'Key design principles', the dictionary data type constitutes an extensive base for future versions of HydroUnits which are expected to support context-based dimensional analysis.
(b) Because even the 10-tuple vectorial representation for the base units are programmatically computed, we believe that it would require very little effort to adjust the vectorial output in order for HydroUnits to accommodate situations where 7-tuple and 8-tuple vectorial representation are needed. This is an aspect of the reusability and scalability traits that we expected HydroUnits to exhibit.
(c) HydroUnits' ability to handle standard-based unforeseen units and its capability to host user-defined units make it very suitable to be used in almost any domain with very minimal alteration. For user-defined units, we foresee the modification of only the 'units' knowledge base module with its easy-to-follow units and conversion factors definition pattern. In other words, no refactoring is necessary. These two core features offer the possibility to accommodate domain-specific units and also new ones that may emerge due to the evolution of science and engineering.

In the future we intend to deploy this tool with an end-to-end (sensor to publication) data acquisition system based on Raspberry Pi (http://www.raspberrypi.org/) and/or Arduino (http://www.arduino.cc/) devices to measure parameters relevant to hydro-meteorological processes such as rainfall, temperature, soil moisture, relative humidity, among others. Our hardware architecture currently comprises a Raspberry Pi that will serially control a Arduino device to add computational power and intelligence such as data quality control and quality assurance, multiple and independent sampling interval, short-term trend detection of variables to trigger related sensor on a slower or faster sampling interval and, more importantly, data product computation such as evapotranspiration. Integration of the HydroUnits package into sensor platform will permit us to compute data products from sensors measuring data, for example, in a different unit system. Ultimately, we would like to develop a system that incorporates the unit conversion system such that the TEDS are automatically decoded for random sensors attached to a data acquisition system (e.g., a Raspberry PI) and then proceeds to automatically select appropriate sensors (and associated unit conversions) to compute added value products such as evapotranspiration. While our development environment is an Intel Core i5 microprocessor-based computer running on Microsoft Windows, HydroUnits has successfully been installed and executed without modification on the single-board microcomputer Raspberry Pi (Model B running Raspbian OS, CPU is an ARM processor with 700 MHz clock speed and RAM is 512 MB). This preliminary experiment demonstrates that HydroUnits can be seamlessly integrated into our envisioned datalogging system. Moreover, this tool can be particularly useful when data integration/fusion and data assimilation from heterogeneous sensors and sensor networks in a hydrologic systems modeling environment becomes necessary.

ACKNOWLEDGEMENTS

This work is currently being supported by the Grove School of Engineering at The City College of New York, the City University of New York's Environmental CrossRoads Initiative and a grant from the IEEE Foundation to develop the envisioned end-to-end data acquisition system. We thank the three reviewers whose comments and suggestions contribute to the improvement of the paper. Special appreciation goes to Matia Joseph and Marie Rose Jean Marie for their support during the course of this software development effort.

REFERENCES

Allen, E., Chase, D., Luchangco, V., Maessen, J. & Steele, G. 2004 Object-oriented units of measurement. *ACM SIGPLAN Notices* **39** (10), 384–403.

Babovic, V. 2009 Introducing knowledge into learning based on genetic programming. *J. Hydroinform.* **11** (3–4), 181–193.

Baldwin, G. 1987 Implementation of physical units. *ACM SIGPLAN Notices* **22** (8), 45–50.

Beazley, D. & Jones, B. K. 2013 *Python Cookbook*, 3rd edn. O'Reilly Media, Sebastopol, CA.

Boggs, J. 1958 The logarithm of 'ten apples'. *J. Chem. Educ.* **35** (1), 30–31.

Brown, W. E. 2001 Applied template metaprogramming in SI units: the library of unit-based computation. In: *Proceedings of the Second Workshop on C++ Template Programming*, Tampa Bay, FL, USA.

Celicourt, P. & Piasecki, M. 2015 An IEEE 1451.0-based platform-independent TEDS creator using open source software components. *Int. J. Sensors Sensor Netw.* **3** (1), 1–11.

Cmelik, R. F. & Gehani, N. H. 1988 Dimensional analysis with C++. *IEEE Softw.* **5** (3), 21–27.

Cunis, R. 1992 A package for handling units of measure in Lisp. *ACM SIGPLAN Lisp Pointers* **V** (2), 21–25.

Damevski, K. 2009 Expressing measurement units in interfaces for scientific component software. In: *Proceedings of the 2009 Workshop on Component-Based High Performance Computing*, Portland, OR, p. 13.

Dreiheller, A., Mohr, B. & Moerschbacher, M. 1986 Programming pascal with physical units. *ACM SIGPLAN Notices* **21** (12), 114–123.

Gehani, N. 1977 Units of measure as a data attribute. *Comput. Lang.* **2** (3), 93–111.

Gehani, N. 1985 Ada's derived types and units of measure. *Softw. Pract. Exper.* **15** (6), 555–569.

Goldberg, D. 1991 What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.* **23** (1), 5–48.

Goldberg, D. 2000 A hydroinformatician's approach to computational innovation and the design of genetic algorithms. *J. Hydroinform.* **2**, 155–162.

Gonzalez, D. & Peart, T. 1993 Applying dimensional analysis. *ACM Sigada Ada Lett.* **XIII** (4), 77–86.

Gruber, T. R. & Olsen, G. R. 1994 An ontology for engineering mathematics. In: *Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, pp. 258–269.

Hamilton, B. 1996 A compact representation of units. Hewlett-Packard Laboratories, Technical Publications Department.

Hilfinger, P. 1988 An Ada package for dimensional analysis. *ACM Trans. Programm. Lang. Syst.* **10** (2), 189–203.

Hillar, G. 2013 Quantities and Units in Python. Dr. Dobb's. http://www.drdobbs.com/jvm/quantities-and-units-in-python/240161101 (retrieved 6 February 2015).

Horsburgh, J. S., Tarboton, D. G., Maidment, D. R. & Zaslavsky, I. 2008 A relational model for environmental and water resources data. *Water Resour. Res.* **44**, W05406.

Horsburgh, J. S., Tarboton, D. G., Schreuders, K. A. T., Maidment, D. R., Zaslavsky, I. & Valentine, I. 2010 HydroServer: A platform for publishing space-time hydrologic datasets. In: *Proceedings of the AWRA Spring Specialty Conference on GIS and Water Resources*, Orlando, FL, pp. 29–31.

House, R. 1983 A proposal for an extended form of type checking of expressions. *Computer J.* **26** (4), 366–374.

Huai, W., Chen, G. & Zeng, Y. 2013 Predicting apparent shear stress in prismatic compound open channels using artificial neural networks. *J. Hydroinform.* **15** (1), 138–146.

IEEE 1997 'IEEE Std 1451.2-1997', IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, 1997.

IEEE 2007 IEEE Standard for a Smart Transducer Interface for Sensors and Actuators–Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Standard 1451.0-2007.

Jiang, L. & Su, Z. 2006 Osprey: a practical type system for validating dimensional unit correctness of c programs. In: *Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, pp. 262–271.

Karr, M. & Loveman, D. 1978 Incorporation of units into programming languages. *Commun. ACM* **21** (5), 385–391.

Kennedy, A. 1994 Dimension Types. In: Programming Languages and Systems-ESOP'94: 5th European Symposium on Programming, Edinburgh, UK, April 11–13, 1994. Proceedings (vol. 788, p. 348). Springer Science & Business Media, Berlin.

Kent, W., Janowski, S. L., Hamilton, B. & Hepner, D. 1996 'Measurement Data (Archive Report)', April 1996. [135 pp]. http://www.hpl.hp.com/techreports/96/HPL-96-73.pdf (accessed February 2015).

Khanin, R. 2001 Dimensional analysis in computer algebra. In: *Proceedings of the 2001 International Symposium on*

Symbolic and Algebraic Computation, London, ON, Canada, pp. 201–208.

Krasner, G. E. & Pope, S. T. 1988 A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *J. Object Orient. Prog.* **1** (3), 26–49.

Krueger, C. 2001 Using separation of concerns to simplify software product family engineering. In: *Proceedings of the Dagstuhl Seminar No. 01161: Product Family Development*. Wadern, Germany.

Liriano, S. & Day, R. 2001 Prediction of scour depth at culvert outlets using neural networks. *J. Hydroinform.* **3**, 231–238.

Männer, R. 1986 Strong typing and physical units. *ACM SIGPLAN Notices* **21** (3), 11–20.

Matta, C., Massa, L., Gubskaya, A. & Knoll, E. 2011 Can one take the logarithm or the sine of a dimensioned quantity or a unit? Dimensional analysis involving transcendental functions. *J. Chem. Educ.* **88** (1), 67–70.

Mills, I. 1995 Dimensions of logarithmic quantities. *J. Chem. Educ.* **72** (10), 954.

Minns, A. 2000 Subsymbolic methods for data mining in hydraulic engineering. *J. Hydroinform.* **2**, 3–13.

Molyneux, P. 1991 The dimensions of logarithmic quantities: implications for the hidden concentration and pressure units in pH values, acidity constants, standard thermodynamic functions, and standard electrode potentials. *J. Chem. Educ.* **68** (6), 467–476.

Morshed, J. & Powers, S. E. 2000 Regression and dimensional analysis for modeling two phase flow. *Transport Porous Media* **38** (1–2), 205–221.

Novak, G. 1995 Conversion of units of measurement. *IIEEE Trans. Softw. Eng.* **21** (8), 651–661.

Oliphant, T. 2007 Python for scientific computing. *Comput. Sci. Eng.* **9** (3), 10–20.

Petty, G. 2001 Automated computation and consistency checking of physical dimensions and units in scientific programs. *Softw. Pract. Exper.* **31** (11), 1067–1076.

Python Software Foundation 2014 Index of Packages Matching 'unit conversion': Python Package Index. https://pypi.python.org/pypi?%3Aaction=search&term=unit+conversion&submit=search (retrieved 6 February 2015).

Reenskaug, T. 2003 The Model-View-Controller (MVC): Its Past and Present. Draft of August 20, 2003. http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf (accessed 15 January 2015).

Reenskaug, T. 2007 The original MVC reports. http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf (accessed 15 January 2015).

Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., Günther, H. M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J. E. H., Singer, L., Fox, R., Weaver, B. A., Zabalza, V., Edwards, Z. I., Bostroem, K. A., Burke, D. J., Casey, A. R., Crawford, S. M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P. L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M. & Streicher, O. 2013 Astropy: a community Python package for astronomy. *Astronomy Astrophysics* **558**, A33.

Rogers, P. 1988 Dimensional analysis in Ada. *ACM Sigada Ada Lett.* **VIII** (5), 92–100.

Saleh, M. & Gomaa, H. 2005 Separation of concerns in software product line engineering. *SIGSOFT Softw. Eng. Notes* **30** (4), 1–5.

Song, E. & Lee, K. 2008 Understanding IEEE 1451-Networked smart transducer interface standard – What is a smart transducer? *IEEE Instrument. Measure. Mag.* **11** (2), 11–17.

Thompson, A. & Taylor, B. N. 2008 Guide for the Use of the International System of Units (SI) NIST Special Publication 811, 2008 edition (version 3.0). http://physics.nist.gov/SP811 [Saturday, 24-Jan-2015 22:46:15 EST]. National Institute of Standards and Technology, Gaithersburg, MD.

Tratt, L. 2009 Dynamically typed languages. *Adv. Comput.* **77**, 149–184.

First received 17 April 2015; accepted in revised form 14 October 2015. Available online 4 December 2015