UnitsML Guidelines Version 1.0

Working Draft 01

[DD Month YYYY]

Abstract:
[Summary of the technical purpose of the document]

Status:
This Working Draft (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or approved as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document Approval Process begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

This document is intended to become a Non-Standards Track Work Product. The patent provisions of the OASIS IPR Policy do not apply.

Table of Contents

List Of Illustrations

List Of Listings

Introduction

In our physical world, units are everywhere. We as humans compare distances, time, weight, value, and easily use a system of scales and references to do so. The grocery store is further away than the workplace. The road bike is lighter than the downhill mountain bike. The time it takes to read these guidelines is longer than the time it takes to not do so.

We as humans are constantly aware of the value of something in reference to something else, and only seldom of its absolute, numerical value. Maybe it is because of this fact, that we do not forget the kind of value that we are talking about. To rehash the examples above, a human would not have the idea of claiming that the grocery store is `bigger' (further away) than (the weight of) the road bike, or that not reading this document is `shorter' than (the weight of) the mountain bike. Actually human language does not prevent us from stating something obviously that wrong, but a layer of inference kicks in to deliver missing or incomplete information.

“The grocery store is further than the road bike [sic!].” Interpreting this sentence leads us to think, “The distance from here to the road bike, wherever it may be, is less than the distance from here to the grocery store. (Let's use the road bike to get to the store then).” This is an example of inference of missing facts from the surrounding haze, correcting wrong input so to speak and making something sensible out of it. In the end, this might not be what whoever said the first sentence wanted to say, and this is where serious misunderstandings can ensue; but then again, we as humans have experience in dealing with erroneous or incomplete information and are very likely to get it right with little effort.

Units are not easy

This small example should have shown though that units of measure and their related concepts (dimensions, quantities, prefixes, unit systems, conversions, derivations, etc.) can be quite confusing and ambiguous. In fact, trying to get `marking up units' right is not an easy task. There are several pitfalls related to different approaches in dealing with units:

- **Code lists** – can contain an arbitrary amount of units and information about them but will need a code after all to describe the units. Usually these code lists are compiled once and then used in scope. They are thus typically static; only a selection of units is represented; the format is fixed.
- **Symbols** – are a light-weight way to `mark up' numerical values with units. But once units from outside the SI come into play, the symbols no longer are unambiguous. Consider `nm' for example: nanometers or nautical miles?
- **Names** – are usually descriptive and well-known to the users. The only problem that remains is that not all names (especially outside the SI) are

> standardized. Consider the pound. Is it a US pound, and thus about 453.6 grams? Or is it maybe a metric pound, and thus exactly 500 grams? Maybe it's a troy pound, or even an avoirdupois pound. All of these are different, but in their domain usually just called “pound”

So communicating units of measure is not as trivial as it might appear at first. There is a lot of information available about units of measure – and a lot of it is required while exchanging numerical data that is related to units of measure.

Computers need help

Given that deducing missing information is not generally a strength of computers nowadays, care has to be taken to add this extra information into the languages being used to have computers communicate amongst each other. For this purpose, there is a need for an unambiguous language to communicate units and all of their (necessary) related concepts. Such a language can be used to enrich the computer-to-computer communication so as to avoid confusion about the currently used units, unit systems, quantities etc.

Another application of such a language has been suggested as early as 1986 by Dreiheller et al. ([DREIHELLER]) Input and output should be performed with proper unit and dimension checks. The authors offer an example of a fatal error caused by the lack of such unit checks on input and output: During the test of the Strategic Defense Initiative (`SDI', colloquially `Star Wars') program in the United States in the mid-80s, the space shuttle Discovery flew over Maui facing in the wrong direction. The cause of the problem was that the computer system on board was not capable of determining input in unexpected units and not displaying the units of measure being used:

> “Much to the surprise of Mission Control, the space shuttle Discovery flew upside-down over Maui on 19 June 1985 during an attempted test of a Star-Wars-type laser-beam missile defense experiment. (...) the experiment failed because the shuttle's reflecting mirror was oriented upward! A statement issued by NASA said that the shuttle was to be repositioned so that the mirror was pointing (downward) at a spot 10,023 feet above sea level on Mona Kea ; that number was supplied to the crew in units of feet, and was correctly fed into the on-board guidance system – which unfortunately was expecting units in nautical miles, not feet. Thus the mirror wound up being pointed (upward) to a spot 10,023 nautical miles above sea level.” ([SIGSOFT], p. 12)

Introduction to Physical Quantities and Scientific Units of Measure

Before getting to discussion details about UnitsML, let's first make a brief detour introducing physical quantities and their relation to scientific units of measure.

One definition of a physical quantity is the measurable property of a thing. Examples of physical quantities are length, mass, and velocity. The value of a quantity is its magnitude expressed as the product of a number and a scientific

unit of measure, and the number multiplying the unit is the numerical value of the quantity expressed in that unit of measure.

Any quantity can be expressed in terms of other quantities through a mathematical representation. It is convenient to define a set of base quantities through which all other quantities, called derived quantities, can be expressed. ISO 31 ([ISO31]) follows this convention and defines seven base quantities: length, mass, time, electric current, thermodynamic temperature, amount of substance and luminous intensity. In the SI, the seven base unit names and symbols used for expressing values of the seven base quantities are given in Table 1 below.

Base Quantity	Base Unit	
	Name	**Symbol**
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd

Table 1: Seven base quantities and their corresponding SI base unit names and symbols

There is one common usage of expressing the relationship between quantities and units that is technically incorrect and can lead to confusion. Frequently, an aspect of a physical quantity is treated as if it is a unit of measure. For example, the expression “emission rate = 1.36 e/s”, where `e' represents electron, treats `electron' as a unit. The correct expression should be “electron emission rate = 1.36 s^{-1}”, or “electron emission rate = 1.36 /s”. Please refer to the discussion in Counted Items on page 22 for more information about how UnitsML treats this issue.

Purpose of UnitsML

Units Markup Language (UnitsML, [UNITSML]) is a markup language for encoding scientific units of measure in XML. The language development was initially started as a project at the National Institute of Standards and Technology, where further accompanying components keep being developed until today (a database

containing detailed information on SI (International System of Units (Système International d'Unités)) and

non-SI scientific units of measure, and tools to facilitate the incorporation of UnitsML into other markup languages). The language development process has switched over to the Organization for the Advancement of Structured Information Standards (OASIS) in June 2006 – cf. Development Process of UnitsML on page 9.

The availability of a markup language for units allows for the unambiguous storage, exchange, and processing of numerical data, thus facilitating the collaboration and sharing of information over the Internet. It is anticipated that UnitsML will be used by the developers of other markup languages to address the needs of specific communities (e.g. mathematics, chemistry, materials science, business/commerce, etc.). Use of UnitsML in other markup languages will reduce duplication of effort and improve compatibility among specifications that represent numerical data.

The XML schema for UnitsML allows for the ability to represent scientific units of measure in XML and will be used for validating XML documents that use UnitsML. The UnitsML schema can be used as a building block for other markup languages by embedding a subset of the language into another markup language. It can as well be used to remain complete and intact and coexist with other markup languages in a single document. And finally it can be used to communicate facts about units of measure all on its own.

SI units can be represented through the use of base units (e.g., meter, second), special derived units (e.g., joule, volt), and any combination of these units with appropriate prefixes and exponential powers (e.g., $mm \bullet s^{-2}$). Aside of creating units necessary for the user's domain of application, commonly used derived SI units (e.g., square meter, meter per second) and non-SI units (e.g., minute, ångström, and inch per second) will be explicitly supported for reference within XML documents.

Development Process of UnitsML

UnitsML has been developed within a technical committee ([UNITSML]) of the Organization for the Advancement of Structured Information Standards (OASIS) and will be released as a standard to the public after the due feedback rounds and running through the OASIS-internal processes. Due to the bylaws of the OASIS, there must not and cannot be patents pending or otherwise impeding the usage or further development of the standard itself.

References

DREIHELLER: A. Dreiheller and M. Moeschbacher and B. Mohr, Programming Pascal with Physical Units, 1986

SIGSOFT: Peter G. Neumann, Risks to the public in computer systems, 1985

ISO31: ISO Technical Committee ISO/TC12, Quantities and units, ISO Standards handbook, 1993

UNITSML: Robert Dragoset, OASIS UnitsML technical committee, 2011, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=unitsml

VIM: International Organization for Standardization, International vocabulary of metrology -- Basic and general concepts and associated terms, 2008

WSDL: , Web Services Description Language (WSDL), , http://www.w3.org/TR/wsdl

SP811: Thompson, Ambler; Taylor, Barry N., Guide for the Use of the International System of Units (SI). NIST Special Publication 811, 2008

SP330: , The International System of Units (SI); NIST Special Publication 330, 2008

XSD1: , XML Schema Part 1: Structures,

XMLID: , xml:id Version 1.0 -- W3C Recommendation 9 September 2005, 2005, http://www.w3.org/TR/xml-id/

XMLNAMES11: , Namespaces in XML 1.1,

The UnitsML Language

The UnitsML language was designed to be embeddable into other markup languages to describe the units of measure being used as well as to stand on its own. If it is embedded into another language, that language shall be referred to here as the "target language".

UnitsML supports robust description of units, items, quantities, dimensions and prefixes. The following sections shall describe the UnitsML language in more detail, starting with a big picture of how UnitsML instance documents are structured, followed by details about each of the critical elements of the language.

The normative reference to UnitsML though is solely the W3C XML Schema, which is published on the homepage of the OASIS technical committee ([UNITSML]). It is reproduced for convenience in the appendix of this document (see page XX Note: TBD).

The Big Picture

UnitsML consists of up to five different sets of elements describing various aspects of units of measure: **<UnitSet>**, **<CountedItemSet>**, **<QuantitySet>**, **<DimensionSet>** and **<PrefixSet>**. Not all of these have to be present, but typically at least one of the **<UnitSet>** or the **<CountedItemSet>** will be present, accompanied by additional information in a **<QuantitySet>**, **<DimensionSet>** and/or **<PrefixSet>**.

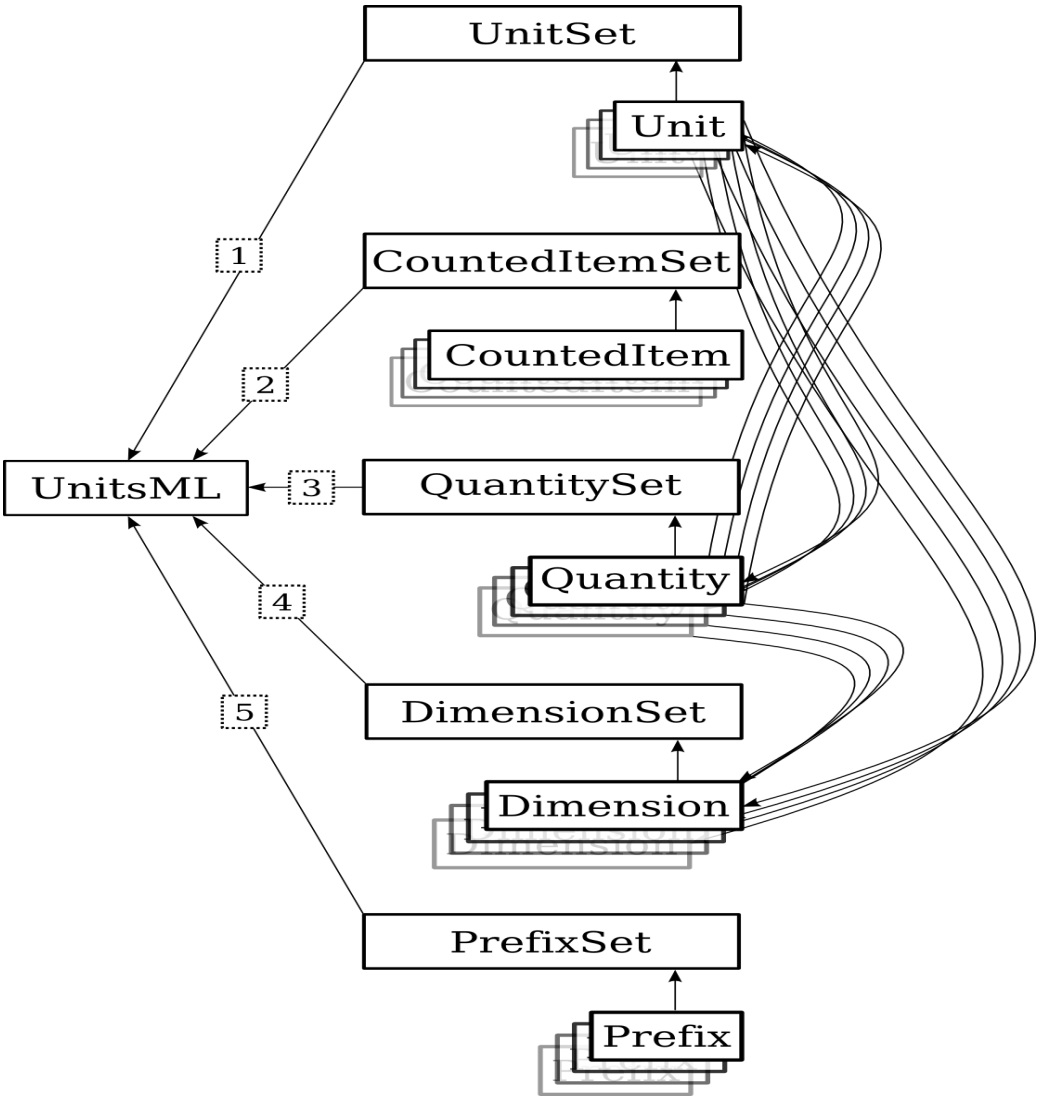

Illustration 1: UnitsML Overview

Each of these **<*Set>**s can contain potentially multiple **<Unit>** (**<CountedItem>**, ...) elements which describe the units of measure used in the instance documents. Depending on the needs of the target language, including one or more of these sets may be enough to allow for marking up the used entities.

The units of measure used (or counted items) typically refer to information about their quantity, dimensionality and so on – this information does not have to reside in the same element though. These references take the form of an URI, which can be resolvable to UnitsML content. The elements which can be referenced in this manner carry an **xml:id** attribute.

Units

What are units?

A measurement unit is a “real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number" ([VIM], section 1.9, page 6).

Or in other words, there exists a reference scale of the same kind of quantity as the quantity to be measured. This scale can then be used to express the magnitude of the phenomenon in question. This shows how tightly units of measure are integrated with the concept of the quantity (see also What is a quantity? on page 24). This tight coupling between units and quantities also shows in the definition of the quantity itself:

> a “property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference
>
> [. . .]
>
> Note 2 : A reference can be a measurement unit, a measurement procedure, a reference material, or a combination of such.” ([VIM], section 1.1, page 2)

Relation to quantities and dimensions

Units are samples from a specific quantity, chosen to measure phenomena of the same kind of quantity as the sample taken. Due to this definition ([VIM], section 1.9, page 6), the quantity from which the unit was chosen and the quantity being measured must be of the same kind (in the sense of the International Vocabulary of Metrology ([VIM])). Given the fact that a single unit can be related to multiple quantities (and those not necessarily having the same kind), it is necessary to retain the information of which exact quantity the reference sample (i.e., the unit in question) was taken. Thus, in UnitsML, a unit of measure should contain a reference to its encompassing quantity (via the **`<QuantityReference>`** element).

Due to the relation of quantities and dimensions (see also What is a dimension? on page 28), a unit can be said to have a given dimension if that dimension matches the quantity dimension of the unit's related quantity. To model this in UnitsML, units of measure can also contain a pointer to a dimension[1] via the **`dimensionURL`** attribute on the **`<Unit>`** element.

1 A unit of measure can be taken from a quantity whose quantity dimensions are similar to the quantity dimensions of quantities which are not of the same kind. In other words, the dimension that a unit is related to is not unique if its ratio of powers of quantity dimensions is not reduced.

Deriving units

Given that units just represent a fitting sample of a specific quantity, deriving units essentially is the same as deriving quantities. So there is a need for a unit to measure, say, speed, and to create its unit the quotient of length per time is being formed. The SI units of choice for these two base quantities are the meter and the second, thus the unit to express speed is the meter per second, or $\frac{m}{s}$.

The quantity dimension of a given quantity, i.e., the seven-tuple of powers of the base quantities corresponds to the resulting (reduced) powers of the units of the involved base quantity. Thus the dimensioning of speed is Length per Time (due to the derivation of the quantity, length per time) which is mirrored in the unit of speed $m^{1} \bullet s^{-1}$.

Prefixing units

When prefixing a unit, a multiplier is being applied on the number expressing the relation to the unit in question. This multiplier can also imply a certain method or range of measurement. Consider e.g. that the numbers are the same for a speed measured in kilometers per second and a speed measured in meters per millisecond.

Prefixing units is similar to deriving a unit from itself (with regard to the chosen reference phenomenon of the chosen quantity) while changing the applied scale of the reference at the same time. Thus, in UnitsML to create a prefixed unit, it is advisable to derive and declare such units from their unprefixed version.

The **<Unit>** element

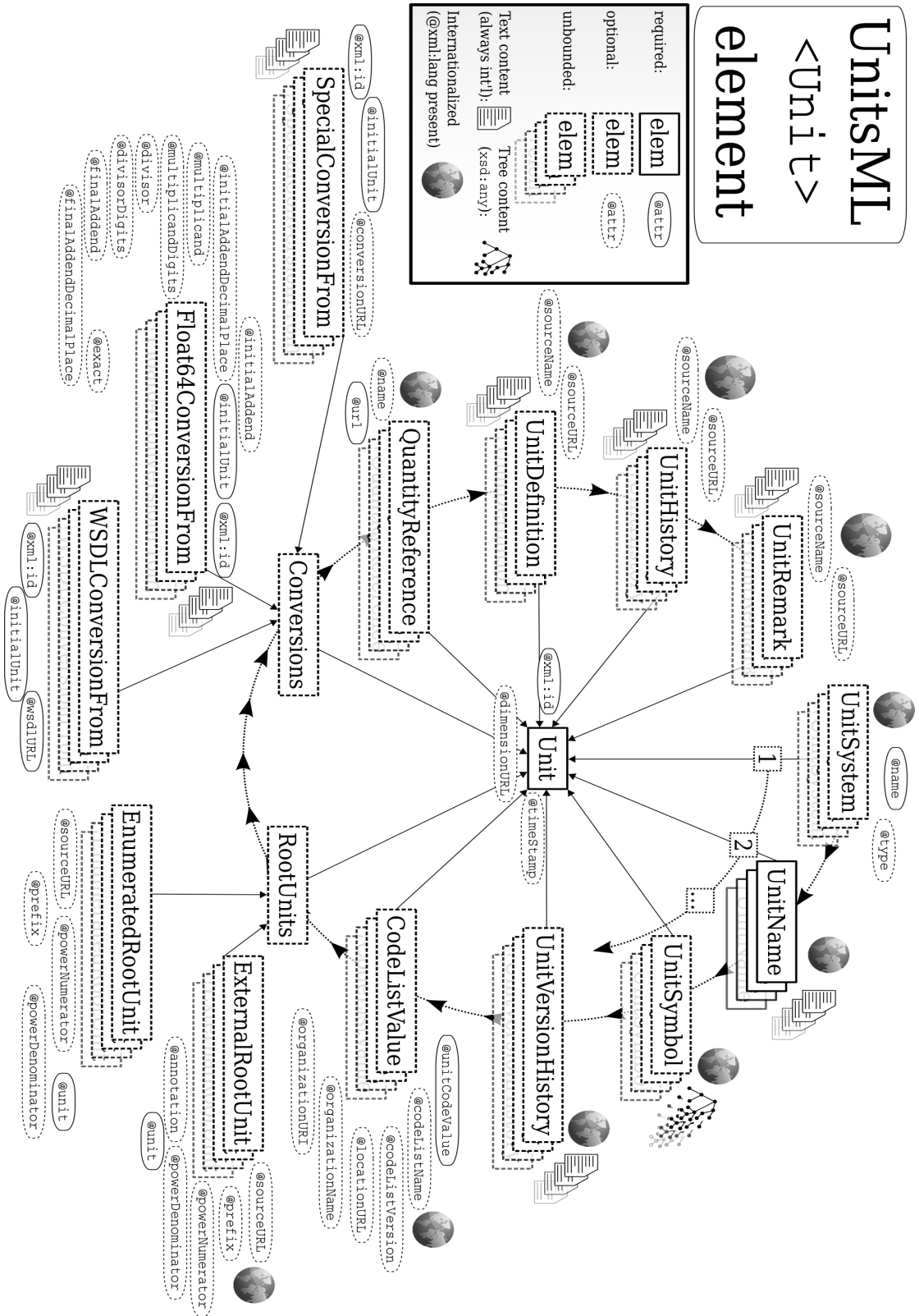

Illustration 2: The UnitsML <Unit> element

The **<Unit>** element is by far the most complex of the elements of UnitsML, mostly because of the information necessary to describe how the unit in question is derived from other units of measure (**<RootUnits>**), how the unit may be abbreviated due to given code lists (**<CodeListValue>**) and how numerical values related to these units can be converted to other units of measure (**<Conversions>**). Each of these elements will be described in more detail. Aside from this extra information, most elements of **<Unit>** allow marking up text with a specified meaning.

Units need to be addressable, thus they carry an **xml:id** attribute. Their **dimensionURL** attribute allows addressing a given dimension directly instead of doing so via the related quantities. Finally the **timeStamp** attribute allows keeping track of different versions of the UnitsML data representing a given unit of measure.

The most visible information about a unit of measure probably is its name and symbol. This information can be stored in the **<UnitName>** and **<UnitSymbol>** child elements, respectively. The **<UnitName>** element is an internationalized (carries an **xml:lang** attribute) text container for the unit name, whereas the **<UnitSymbol>** element can contain any valid XML-based structured markup (or just plain text) as well. Along with carrying an **xml:lang** attribute for the symbol, the required attribute **type** (such as HTML, LaTeX, MathML) must be supplied to describe which markup (which is not necessarily XML or SGML) is being used in the symbol.

As detailed above (Relation to quantities and dimensions on page 12) units of measure are related to quantities, potentially many. This information can be stored with the **<QuantityReference>** child element, whose **url** attribute is required (note that certain tools may expect the referenced content to resolve to valid UnitsML). For declarative purposes this reference can be named (**name** attribute) and internationalized (**xml:lang** attribute).

Unit meta-data in UnitsML

The following **<Unit>** child elements carry meta information about the unit in question:

- **<UnitSystem>** describes the unit system (cf. [[VIM], section 1.13, page 8]) in which the unit resides (which need not be unique, i.e., a unit may reside in multiple unit systems). The system can be described by its name and, optionally, type . It is internationalized (carries an **xml:lang** attribute).
- **<UnitVersionHistory>** contains descriptive information for the historic development of the UnitsML representation of the unit. It is internationalized (carries an **xml:lang** attribute).
- **<UnitDefinition>** details how the unit of measurement itself is being dened by the appropriate standards body. It can contain text as well as reference the appropriate source of the definition via name and URL (the

sourceName and **sourceURL** attributes). It is internationalized (carries an **xml:lang** attribute).

- **<UnitHistory>** contains descriptive information for the historic development of the unit itself . It enables addressing external sources for this (via the **sourceName** and **sourceURL** attributes) next to holding text content. It is internationalized (carries an **xml:lang** attribute).
- **<UnitRemark>** serves as a placeholder for further additional information. Like **<UnitDefinition>** and **<UnitHistory>**, external publications can be referenced by the **sourceName** and **sourceURL** attributes. It is internationalized (carries an **xml:lang** attribute).

Derived Units in UnitsML

Derived units can be modeled in UnitsML with the use of the **<RootUnits>** element. Its possible children are the **<EnumeratedRootUnit>** and **<ExternalRootUnit>** elements. As the name implies, the **<EnumeratedRootUnit>** element gives access to an enumerated list of common root units, which can be used to create new derived units by using these `popular' units. This fixed list of typical root units also improves interoperability by providing a fixed set of well understood units. The **<ExternalRootUnit>** can be used to create new derived units from UnitsML content.

Enumerated root units

<EnumeratedRootUnit> consists of a collection of attributes detailing the derivation. The enumerated list of units is available through the **unit** attribute, the only required attribute of the element. It contains obvious choices, such as `second' or `meter', as well as specialized units such as `printers pica' or `coulomb'. This unit can be combined with a **prefix** attribute, which contains an enumeration of the SI and IEC prefixes (i.e., the well-defined decimal and binary prefixes). To express cases where the unit is included in the expression with an exponent, the attributes **powerNumerator** and **powerDenominator** are offered to form a rational exponent. Finally for documentary purposes, the **sourceURL** attribute can be used to point to a relevant URL for information about the unit used. Listing 1: Sample derived unit using <EnumeratedRootUnit> shows a fragment for how to derive a unit for acceleration, miles per millisecond squared.

```
<UnitsML>
 <!-- ... -->
 <Unit xml:id="u331">
  <!-- ... -->
  <RootUnits>
   <EnumeratedRootUnit unit="mile" />
   <EnumeratedRootUnit prefix="milli" powerNumerator="-2" unit="second" />
  </RootUnits>
 </Unit>
 <!-- ... -->
</UnitsML>
```

Listing 1: Sample derived unit using <EnumeratedRootUnit>

'External' root units

<ExternalRootUnit> gives you full control over the units from which you are deriving. In contrast to **<EnumeratedRootUnit>**, which gives a finite choice of units to derive from, **<ExternalRootUnit>** allows referring to root units via a URL (which is expected but not necessarily resolving to UnitsML content). Its unit attribute thus contains a URI. Next to the **prefix**, **powerNumerator**, **powerDenominator** and **sourceURL** attributes, which have the same purpose as in the **<EnumeratedRootUnit>**, it additionally offers the **annotation** attribute for optionally documenting the unit involved as well as an **xml:lang** attribute for internationalization purposes.Listing 2: Sample derived unit using <ExternalRootUnit> shows a fragment deriving from a counted item (pages) to form the derived unit `pages per hour'.

```
<UnitsML>
 <!-- ... -->
 <Unit xml:id="u337">
  <!-- ... -->
  <UnitName xml:lang="en-US">pages per hour</UnitName>
  <!-- ... -->
  <RootUnits>
   <ExternalRootUnit unit="#i42" annotation="pages" xml:lang="en-US" />
   <EnumeratedRootUnit powerNumerator="-1" unit="hour" />
  </RootUnits>
 </Unit>
 <!-- ... -->
 <CountedItem xml:id="i42">
  <!-- ... -->
  <ItemName xml:lang="en-US">page</ItemName>
  <!-- ... -->
 </CountedItem>
 <!-- ... -->
</UnitsML>
```

Listing 2: Sample derived unit using <ExternalRootUnit>

Referencing code lists in UnitsML

Code list entries can be referenced (or assembled from) the **<CodeListValue>** child element(s). The sole required attribute for a **<CodeListValue>** is its

unitCodeValue attribute. The code list can be named and referred to via the **codeListName** and **locationURL** attributes. Its version can be marked up with the **codeListVersion** attribute. The organization that publishes the given code list can be referred to via its URI (the **organizationURI** attribute) and its name (**organizationName**). Finally **<CodeListValue>**s are also internationalized and thus carry an **xml:lang** attribute (see Listing 3: Sample <CodeListValue> fragment).

```
<Unit>
 <!-- ... -->
 <UnitName xml:lang="en-US">nautical mile</UnitName>
 <UnitName xml:lang="de-DE">Seemeile</UnitName>
 <!-- ... -->
 <CodeListValue unitCodeValue="NM" codeListVersion="2010-03-22"
  codeListName="Aquatic Companies Metric Entities (ACME)"
  locationURL="http://www.acme.com/codes/acme/2010-03-22/"
  organizationName="ACME" organizationURI="urn:megacorp:disney:acme"
  xml:lang="en-US"/>
 <!-- ... -->
</Unit>
```

Listing 3: Sample <CodeListValue> fragment

Unit conversions in UnitsML

Conversions between different units are natively supported in UnitsML by the **<Conversions>** child element of the **<Unit>** element. The **<Conversions>** element is a container for all the possible conversions of a unit. There are three different mechanisms supported for conversions: linear conversions, `special' conversions and conversions that are handled by a web-service.

Linear conversions

Linear conversions between two distinct units are covered by the **<Float64ConversionFrom>** element. Linear conversions are of the form

$y = \frac{(x+a)b}{c}c+d$ where y is the value of the target (current) unit, x the value of the source (`from') unit, and a, b, c, d the parameters of the conversion, called **initialAddend** (a), **multiplicand** (b), **divisor** (c) and **finalAddend** (d) respectively.

Additionally the **<Float64ConversionFrom>** element also supports documenting the decimal place to which the parameters are known to be signicant (e.g., whether a conversion factor is known to be 0.5 or 0.500). For this purpose the attributes **initialAddendDecimalPlace**, **multiplicandDigits**, **divisorDigits** and **finalAddendDecimalPlace** are present on the **<Float64ConversionFrom>** element.Listing 4: Sample linear conversion demonstrates most of the parameters of the **<Float64ConversionFrom>** element with an absolute thermodynamic temperature conversion.

If a conversion is known to be exact (e.g., does not involve π) it can be documented using the **exact** attribute of the conversion. This information does

not refer to whether the actual machine implementation of floating point numbers used can produce an accurate result, but rather whether the conversion itself is known to be exact.

Finally, a **<Float64ConversionFrom>** can also carry a **<ConversionNote>** element to document specific details about a given conversion. This **<ConversionNote>** element can contain plain text or any valid XML markup, and communicate its content's language via the **xml:lang** attribute.

```
<UnitsML>
 <!-- ... -->
 <Unit xml:id="u23">
  <!-- ... -->
  <UnitName xml:lang="en-us">degrees celsius</UnitName>
  <!-- ... -->
  <Conversions>
   <Float64ConversionFrom xml:id="u23_from_u5" initialUnit="#u5"
    finalAddend="-273.15" exact="true" />
   <Float64ConversionFrom xml:id="u23_from_u314" initialUnit="#u314"
    initialAddend="-32" multiplicand="9" divisor="5" exact="true" />
   <!-- divisor="1.8" would have been as good -->
   <!-- ... -->
  </Conversions>
  <!-- ... -->
 </Unit>
 <Unit xml:id="u314">
  <!-- ... -->
  <UnitName xml:lang="en-us">degrees fahrenheit</UnitName>
  <!-- ... -->
 </Unit>
 <Unit xml:id="u5">
  <!-- ... -->
  <UnitName xml:lang="en-us">kelvin</UnitName>
  <!-- ... -->
 </Unit>
 <!-- ... -->
</UnitsML>
```

Listing 4: Sample linear conversion

UnitsML stores information about conversions on the target unit, not on the source units (hence the name, **Float64ConversionFrom**). Whether conversion information is stored transitively is up to the author of the units markup. E.g., one could store conversions from inches to feet and from inches to meters as well as conversions from feet to meters and vice versa. On the other hand it is also possible to compute most of this information with just having the parameters for each single step. For this example this means that if there is conversion information stored for converting from inches to feet, and from feet to meters, then it is trivial to also compute how to convert from meters to feet, feet to inches, inches to meters or meters to inches. For the example in Listing 4, there is enough information to come up with the conversion parameters from kelvin to degrees Fahrenheit. The choice on the scope of documenting these parameters is up to the content author.

Special conversions

Special conversions are those that cannot be easily described as a linear conversion. Instead of containing parameters to a given function, a **<SpecialConversionFrom>** contains information about how to convert from a given unit either as textual documentation or in an alternative valid XML language (e.g., MathML). This can be used freely to model anything from non-linear equations to arbitrary equations in a domain language being used in the environment of the author up to simple textual descriptions of how to convert between given units.

This information is presented in the **<SpecialConversionFrom>**'s **<ConversionDescription>** children, which can be marked with an **xml:lang** attribute to denote the used (natural) language in the conversion's description, and can contain either plain text or any valid(ating) XML markup. An example is given in Listing 5: Sample special conversion.

```
<UnitsML>
 <!-- ... -->
 <Unit xml:id="u23">
  <!-- ... -->
  <UnitName xml:lang="en-us">degrees celsius</UnitName>
  <!-- ... -->
  <Conversions>
   <!-- ... -->
   <SpecialConversionFrom xml:id="u23_from_u314_special"
    initialUnit="#u314">
    <ConversionDescription xml:lang="de">
     subtrahiere 32 vom nummerischen Wert der Grad Fahrenheit
     und teile daraufhin durch 1.8
    </ConversionDescription>
    <ConversionDescription><![CDATA[
    // scala code:
    def fahrenheitToCelsius(f: Double): Double = (f-32)/1.8
    ]]></ConversionDescription>
   </SpecialConversionFrom>
   <!-- ... -->
  </Conversions>
  <!-- ... -->
 </Unit>
 <Unit xml:id="u314">
  <!-- ... -->
  <UnitName xml:lang="en-us">degrees fahrenheit</UnitName>
  <!-- ... -->
 </Unit>
 <!-- ... -->
</UnitsML>
```

Listing 5: Sample special conversion

Conversions by a Web Service

The final way to describe unit conversions within UnitsML is to refer to a web-service doing the actual conversion through the **<WSDLConversionFrom>** element. This element contains a **wsdlURL** attribute to refer to a Web Services Desciption Language ([WSDL]) instance document. Additionally a **<WSDLDescription>** child is supported to document further details or documentation about the function of

the web service. This description is internationalized via the **xml:lang** attribute, and can contain either textual or valid XML markup content (see Listing 6: Sample WSDL conversion).

```
<UnitsML>
 <!-- ... -->
 <Unit xml:id="u23">
  <!-- ... -->
  <UnitName xml:lang="en-us">degrees celsius</UnitName>
  <!-- ... -->
 </Unit>
 <Unit xml:id="u314">
  <!-- ... -->
  <UnitName xml:lang="en-us">degrees fahrenheit</UnitName>
  <!-- ... -->
  <Conversions>
   <!-- ... -->
   <WSDLConversionFrom xml:id="u23_to_u314_wsdl"
    initialUnit="#u23"
    wsdlURL="http://chalk.coas.unf.edu:8080/axis2/services/tempConvert?wsdl">
    <WSDLDescription xml:lang="en-US">
     Example of a WSDL temperature conversion, converting from kelvin
     or degrees celsius to degrees fahrenheit. Parameters, as described
     in the WSDL, too, are the numerical *value* and the source *unit*.
     Example:
     http://chalk.coas.unf.edu:8080/axis2/services/tempConvert/getKelvin?
alue=25&unit=C
    </WSDLDescription>
   </WSDLConversionFrom>
   <!-- ... -->
  </Conversions>
 </Unit>
 <!-- ... -->
</UnitsML>
```

Listing 6: Sample WSDL conversion

Counted Items

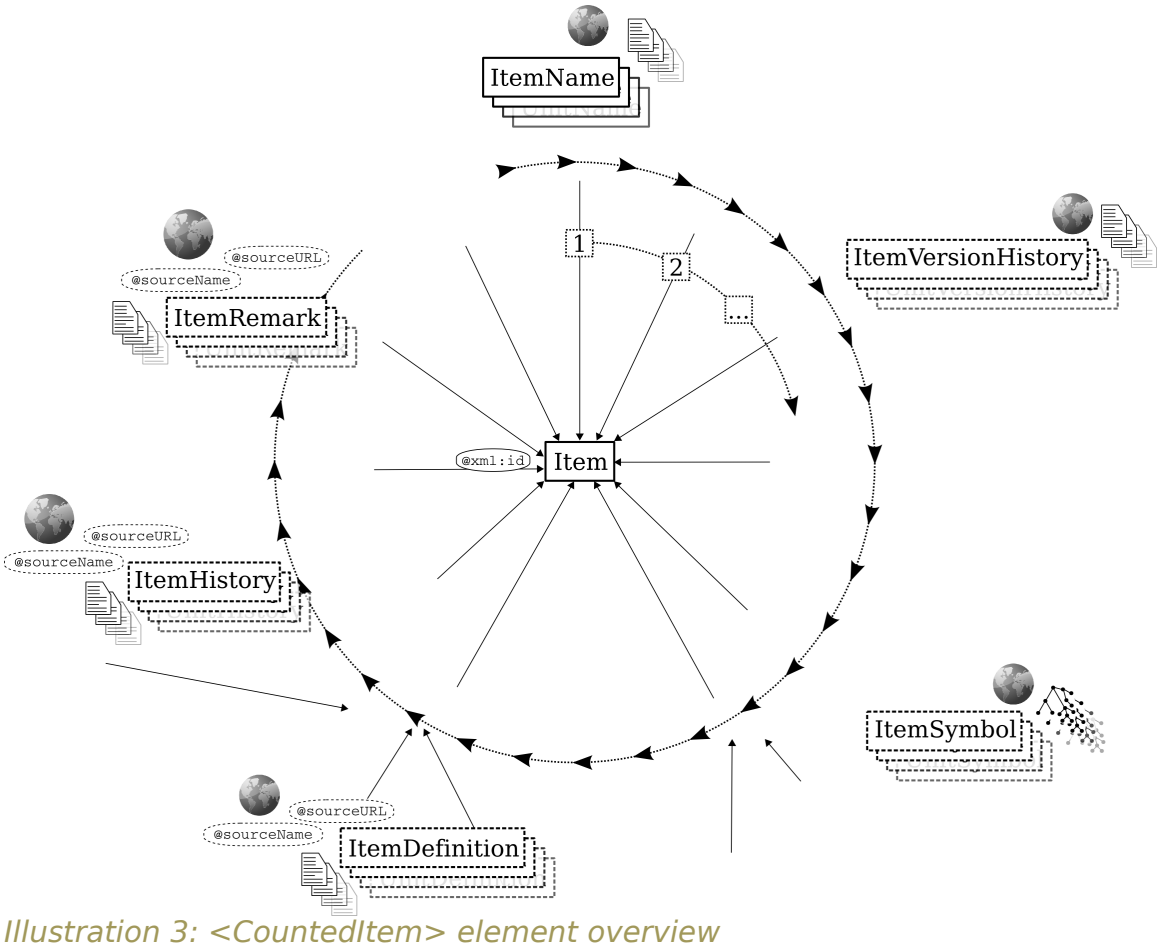

Illustration 3: <CountedItem> element overview

What are counted items?

Counted items really are countable entities which are not units of measure. They are often combined with units of measure in the real world, e.g., if there is a count per time. It is often not obvious whether these counts should be on the unit side of the picture or on the quantity side. As an example consider the following, if we are talking about electron ow per second, would this be a quantity of `electron flow ' with a unit of $s^{-1}$ or a quantity `flow ' with a unit of electrons/s?

The distinction between counted items and units gets even more blurred when consulting the VIM: “Numbers of entities are quantities of dimension one" ([VIM], Note 4, section 1.8, page 6) and its definition of measurement unit : “real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of two quantities as a number" ([VIM], section 1.9, page 6).

In other words, if it is conventional to use a counted item as a reference for a given quantity of counted entities, then this qualifies it as a measurement unit.

On the other hand, the (admittedly not universal) guide for the use of the International System of Units contrasts this with ([SP811], section 7.5, page 17):

> “When one gives the value of a quantity, any information concerning the quantity or its conditions of measurement must be presented in such a way as not to be associated with the unit. This means that quantities must be dened so that they can be expressed solely in acceptable units (including the unit one [...])"

One of the examples given is “the sensitivity for NO_3 molecules is 5 x 10^{10} /cm^3 but not: the sensitivity is 5 x 10^{10} NO_3 molecules / cm^3" (ibid.). UnitsML does not dictate to take either way. For some applications it makes more sense to model a counted item as a measurement unit instead, using the **<Unit>** element, for others, the use of the **<CountedItem>** element may be more appropriate.

The <CountedItem> element

The **<CountedItem>** element can be used to mark up instances of counted entities to go along with a unit expression. This practice is strongly discouraged within the SI (cf. for example [SP811], section 7.5, page 17) although often used in reality. One such example was given earlier already: instead of using an expression like electron flow = *n*/s often the discouraged version `flow = *n* electrons/s' is being used. Furthermore, in contrast to the scientific community, these kinds of `equations' are often used in commerce. In acknowledging this discouraged practice, the **<CountedItem>** element has been added to the UnitsML language.

<CountedItem>s have to carry an **xml:id** attribute, so they can be referenced from elsewhere within a UnitsML or UnitsML-extended instance document. The child elements of the **<CountedItem>** carry mostly textual information referring to the item in question. The sole required child is the name of the item, **<ItemName>**, which can be localized with the **xml:lang** attribute. If the item in question is to appear in equations, a symbol should be added via the **<ItemSymbol>** element. Like the symbols in units and quantities, these can be localized via the **xml:lang** attribute but also have to carry a **type** attribute, determining the type of markup that is being used. The **<ItemSymbol>** element can have any simple (string) or complex (tree) content, as long as it is well-formed XML.

The other child elements of **<CountedItem>** are for informational purposes only and not expected to be interpreted further by software supporting UnitsML. All of these child elements can be localized via the **xml:lang** attribute.

<ItemDefinition>, **<ItemHistory>** and **<ItemRemark>** additionally can reference a source document and/or location containing further information about its definition, history and remarks about the item in question itself. These references are accomplished via the **sourceName** and **sourceURL** attributes.

A complete example of the counted item element is provided in Listing 7: Sample <CountedItem> element.

```
<CountedItem xml:id="i42">
 <ItemName xml:lang="en-US">12 oz. bottle</ItemName>
 <ItemVersionHistory xml:lang="en-US">added "12 oz. " to the item's
name</ItemVersionHistory>
 <!-- no symbol! -->
 <ItemDefinition xml:lang="en" sourceName="Anheuser-Busch company
definitions"
  sourceURL="http://www.anheuser-
busch.com/imaginary/definitions/12ozbottle.html">
  The 12 fluid ounces bottle as being used by the Anheuser Busch companies
 </ItemDefinition>
 <ItemHistory xml:lang="en" sourceName="Anheuser-Busch chronicles"
  sourceURL="http://www.anheuser-
busch.com/imaginary/chronicles/12ozbottle.html" />
 <ItemRemark xml:lang="en" sourceName="Anheuser-Busch feedback"
  sourceURL="http://www.anheuser-
busch.com/imaginary/feedback/12ozbottle.html">
  Note that the european market prefers a different bottle size!
 </ItemDefinition>
</CountedItem>
```

Listing 7: Sample <CountedItem> element

Quantities

What is a quantity?

The term *quantity* sadly is a wildly overloaded one in the realm of metrology. While quantities are that specific thing you measure, quantities are also equivalence classes of specific quantities. A quantity can be a single measurement, a reference, a count, … and at the same time an abstract concept, a base quantity, a derived quantity, etc.

From the point of view of UnitsML, quantities shall be as defined in the VIM:

> “[A] quantity [is the] property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference.
>
> Note 1 The generic concept “quantity” can be divided into several levels of specific concepts" ([VIM], section 1.1, page 2)

An example of these different levels of abstraction is given by the quantity *length*, which can e.g. be a radius or a wavelength, or, more specific, a radius of a circle A, or a wavelength of the sodium D radiation.

In other words, quantities can be seen as sets which contain various measured phenomena with a reference, and these references are units of measure. Not all of these references are sensible units of measure though – e.g., there could be a quantity `width of a (specific) table ' whose reference unit is its height, and another one whose reference unit is the meter. From the point of view of

UnitsML, we consider these quantities, that have standardized units of measures as their reference.

Quantities have been demonstrated above to carry different levels of abstraction. Orthogonally to that, quantities can also be split into base quantities and derived quantities. In fact the metric system is based on a unit system which is related to the International System of Quantities – that is consisting of the quantities length, mass, time and so on ([VIM], section 1.6, note 2, page 4).

Relation to dimensions and units

Quantities are an intermediary concept between units of measure and their (quantity) dimensions. Relating to dimensions and units of measure, a quantity will have a single dimension associated with it (cf. What is a dimension? on page 28), which is dependent on the quantity's relation to its base quantities.

Because of the possibility of dimensions containing quantities which are ratios of similar quantities (i.e., those whose dimensioning cancel out each other, for example a mass ratio per time), a single dimension can be base to multiple quantities, so there is a 1:*n* relation between quantities and dimensions.

The relation between quantities and units of measure is more complicated. The same quantity can be measured with different references, i.e., units of measure. There can be a *length* in meters, feet, inches, yards, fathoms et cetera. These could be considered different quantities, although they have a close relation, which is called `*kind of quantity*'. Quantities of the *same kind* refer to the same sort of phenomenon being measured, but they do not have to have the same reference (i.e., unit of measure). Quantities which could be considered `similar' or `the same' thus could have multiple units of measure associated with it. Similarly units of measures can be the reference of multiple quantities, even of those which are not as closely related (are not considered to be of the same kind). There is thus a *m*:*n* relation between quantities and units of measure.

UnitsML and '*kind of quantity*'

`*Kind of quantity*' is an equivalence relation between quantities – only values within these quantities are actually comparable. Keep in mind that even if quantities have the same quantity dimensions, they need not be comparable, i.e., denoting the `*same*' quantity after all. Even though this is an important concept, it is simply defined as “[the] aspect common to mutually comparable quantities " ([VIM], section 1.2, page 3).

Noteworthy are also the following two notes from the VIM ([VIM], section 1.2, page 3):

> “Note 1 The division of the concept of “quantity" according to “*kind of quantity*" is to some extent arbitrary.
>
> Note 2 Quantities of the same kind within a given system of quantities have the same quantity dimension . However, quantities of the same dimension are not necessarily of the same kind."

UnitsML does not provide a way to denote the `*same-kindness*' of quantities. As this distinction (or equivalence) depends on the problem domain, this decision is left to the content authors instead.

The **<Quantity>** element

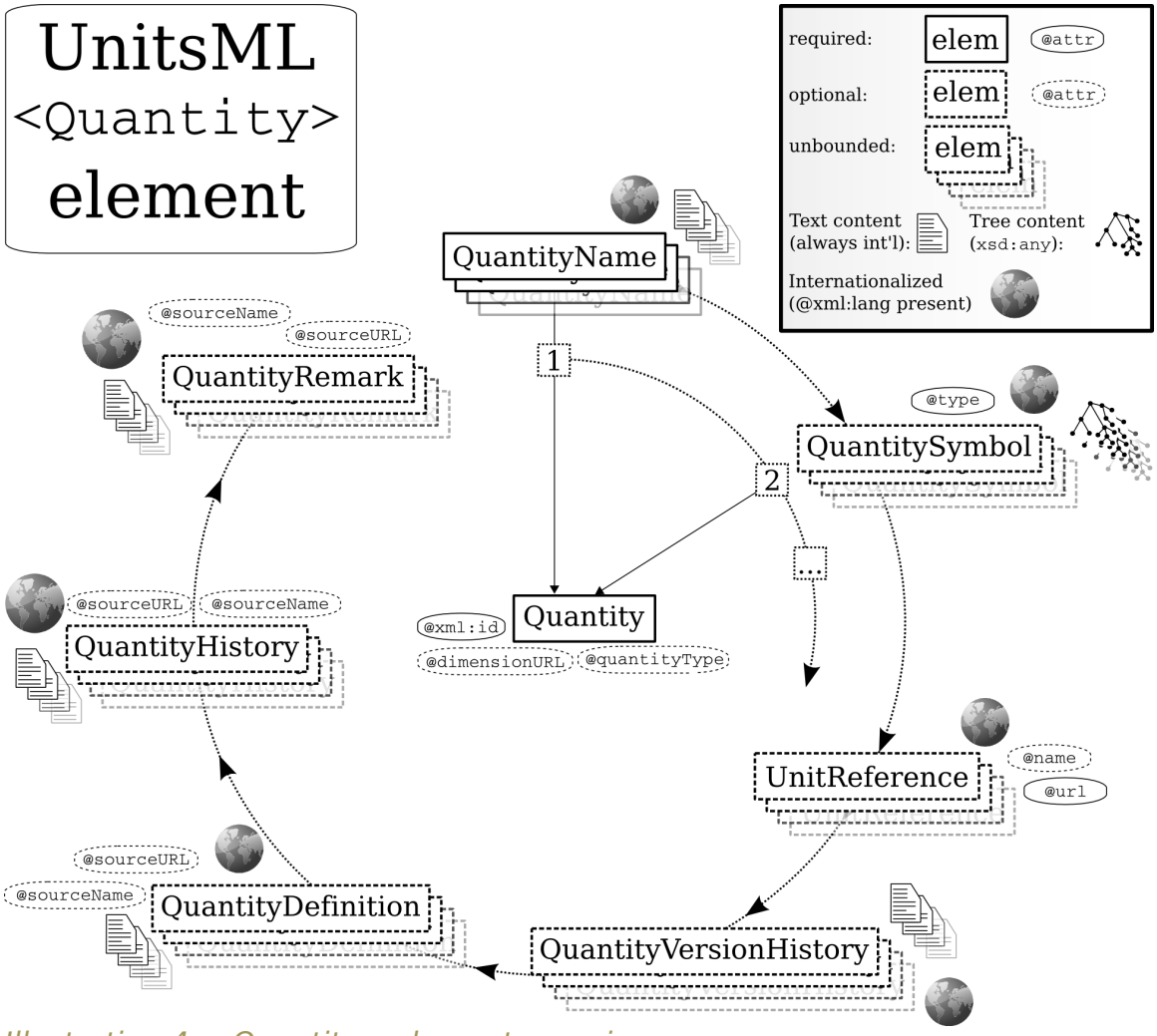

Illustration 4: <Quantity> element overview

Quantities in UnitsML are marked up with the **<Quantity>** element.

The 1:*n* relation between quantities and dimensions is modeled in UnitsML by providing the **dimensionURL** attribute in the **<Quantity>** element. In this way, quantities in UnitsML can refer to a single dimension.

Quantities can carry names via **<QuantityName>** elements, and symbols via the **<QuantitySymbol>** elements. Both are internationalized (i.e., carry an **xml:lang** attribute). **<QuantitySymbol>** additionally has a **type** attribute, which describes the kind of content as it allows any XML markup next to plain text.

In UnitsML, quantities and units of measure can reference each other (multiple times) with the **<QuantityReference>** and **<UnitReference>** child elements of the **<Unit>** and **<Quantity>** elements, respectively. The **<QuantityReference>** element is used to refer to a quantity from a unit whereas the **<UnitReference>** element is used to refer to a unit from a quantity. The **<UnitReference>** element contains a required **url** attribute which ought to refer to the unit of measure, an optional **name** attribute containing the name of the referenced unit, as well as a **xml:lang** attribute for marking up the used language.

The following child elements of **<Quantity>** all describe various meta-data aspects of quantities, similar to that of units of measure (cf. Unit meta-data in UnitsML on page 15):

- **<QuantityVersionHistory>** contains descriptive information for the historic development of the UnitsML representation of the quantity. It is internationalized (carries an **xml:lang** attribute).
- **<QuantityDefinition>** details how the quantity itself is being defined by the appropriate standards body. It can contain text, as well as reference the appropriate source of the definition via name and url (the **sourceName** and **sourceURL** attributes). It is internationalized (carries an **xml:lang** attribute).
- **<QuantityHistory>** contains descriptive information for the historic development of the quantity itself. It enables addressing external sources for this (via the **sourceName** and **sourceURL** attributes) as well as holding textual content. It is internationalized (carries an **xml:lang** attribute).
- **<QuantityRemark>** serves as a placeholder for further additional information. Like **<QuantityDefinition>** and **<QuantityHistory>**, external publications can be referenced by the **sourceName** and **sourceURL** attributes. It is internationalized (carries an **xml:lang** attribute).

An example of two sample **<Quantity>** elements can be seen in Listing 8: Sample <Quantity> element.

```
<!-- ... -->
<QuantitySet>
 <Quantity xml:id="q42" dimensionURL="#d42" quantityType="base">
  <QuantityName xml:lang="en-US">absolute thermodynamic
temperature</QuantityName>
  <QuantitySymbol type="ASCII">T</QuantitySymbol>
  <QuantitySymbol type="LaTeX">{\ensuremath{T}}</QuantitySymbol>
  <UnitReference url="#u5" name="kelvin" xml:lang="en" />
  <UnitReference url="#u23" name="degrees_celsius" xml:lang="en" />
  <!-- ... -->
 </Quantity>
 <Quantity xml:id="q43" dimensionURL="#d42" quantityType="base">
  <QuantityName xml:lang="en-US">thermodynamic temperature
difference</QuantityName>
  <QuantitySymbol type="ASCII">T</QuantitySymbol>
  <QuantitySymbol type="LaTeX">{\ensuremath{T}}</QuantitySymbol>
  <UnitReference url="#u5" name="kelvin" xml:lang="en" />
  <UnitReference url="#u23" name="degrees_celsius" xml:lang="en" />
  <!-- ... -->
 </Quantity>
</QuantitySet>
<!-- ... -->
```

Listing 8: Sample <Quantity> element

Dimensions

What is a dimension?

The term `*dimension*' is clearly defined within the world of the SI. To quote the International Vocabulary of Metrology, where it is more verbosely called "*quantity dimension*": "expression of the dependence of a quantity on the base quantities of a system of quantities as a product of powers of factors corresponding to the base quantities, omitting any numerical factor" ([VIM], section 1.7, page 4), e.g., in the International System of Quantities (of which the SI consists) the quantity dimension of force is denoted by $dim(F)=LMT^2$.

As the definition of the VIM is somewhat abstract, here is the definition from NIST Special Publication 330 ([SP330], excerpt of section 1.3, pp. 11-12):

> "By convention physical quantities are organized in a system of dimensions. Each of the seven base quantities used in the SI is regarded as having its own dimension, which is symbolically represented by a single sans serif roman capital letter. The symbols used for the base quantities, and the symbols used to denote their dimension, are given as follows.
>
> All other quantities are derived quantities, which may be written in terms of the base quantities by the equations of physics. The dimensions of the derived quantities are written as products of powers of the dimensions of the base quantities using the equations that relate the derived quantities to the base quantities. In general the dimension of any quantity Q is written in the form of a dimensional product,

$$dim(Q) = L^{\alpha} M^{\beta} T^{\gamma} I^{\delta} \Theta^{\epsilon} N^{\zeta} J^{\eta}$$

where the exponents α, β, γ, δ, ε, ζ, and η, which are generally small integers which can be positive, negative or zero, are called the dimensional exponents. The dimension of a derived quantity provides the same information about the relation of that quantity to the base quantities as is provided by the SI unit of the derived quantity as a product of powers of the SI base units.

There are some derived quantities *Q* for which the defining equation is such that all of the dimensional exponents in the expression for the dimension of *Q* are zero. [...] Such quantities are described as being *dimensionless*, or alternatively as being of *dimension one* [...].

There are also some quantities that cannot be described in terms of the seven base quantities of the SI at all, but have the nature of a count.[2] Examples are number of molecules [...]. Such counting quantities are usually regarded as dimensionless quantities, or quantities of dimension one, with the unit one, 1."

Relation to quantities and units

From the point of view of the SI, dimensions are not related directly to units of measure. Instead the defining entity is the *quantity*, to which both one (or multiple) units of measure are related, as is its dimensionality. In practice though, as outlined by the quote of the NIST Special Publication 330, the dimensionality is being mirrored in the unit used for a derived quantity: the unit symbols carry the same dimension information as does the construction of the quantity via the base quantities, e.g., velocity consists of *length* per *time*, $V = \frac{l}{t}$. Its dimension thus is $dim(V) = L^1 T^{-1}$ (where dimensions with an exponent of zero have been omitted) which also can be seen from its units, e.g., meters per second: $\frac{m}{s} = m^1 s^{-1}$. So even if units of measure are not directly related to dimensions, the link is easy to construct.

Dimensionality and '*kind of quantity*'

A common approach is to decide on whether values of measured quantities are comparable based on their dimensionality. Remember that this is wrong. The VIM clearly defines when quantities are comparable. For more details, see UnitsML and 'kind of quantity' on page 25.

The <Dimension> element

Dimensions in UnitsML are geared towards the SI's system of quantities, i.e., it provides support for the seven base quantities as outlined in Table 1: Seven base

2 Note that these can be modeled as **<CountedItem>**s in UnitsML

quantities and their corresponding SI base unit names and symbols. Additionally two further factors for the dimensionality of a quantity are offered: *PlaneAngle* and *Item*. The latter is being offered for marking up quantities involving counted quantities, e.g., electrons. Note that neither of these additions are supported by the SI.

The **<PlaneAngle>** is supported in UnitsML for explicitly stating the involvement of an angle in a quantity instead of having to use the (canceling) expression L^nL^{-n}.

```
<Dimension xml:id="dim42">
 <Length powerNumerator="1" powerDenominator="1" />
 <Mass powerNumerator="1" powerDenominator="1" />
 <Time powerNumerator="-2" powerDenominator="1" />
</Dimension>
<!-- dimensionality of energy -->
<Dimension xml:id="dim23a">
 <Length powerNumerator="1" powerDenominator="1" />
 <Mass powerNumerator="1" powerDenominator="1" />
 <Time powerNumerator="-1" powerDenominator="1" />
 <Length powerNumerator="1" powerDenominator="1" />
 <Time powerNumerator="-1" powerDenominator="1" />
</Dimension>
<!-- Note: default of powerDenominator / powerNumerator is 1 -->
<!-- following version thus is equal to the above for schema-aware
     xml processors -->
<Dimension xml:id="dim23b">
 <Length />
 <Mass />
 <Time powerNumerator="-1" />
 <Length />
 <Time powerNumerator="-1" />
</Dimension>
```

Listing 9: Sample <Dimension> elements

This example defines two quantity dimensions, that of force (**dim42**) and that of energy (**dim23a**, **dim23b**, **dim23c**). Note that the order of the **<Length>**, **<Mass>** etc. elements does not matter – each of the child elements of the **<Dimension>** element refers to a factor in the dimensional equation with the given rational exponent. Thus the third definition of energy is equivalent to the first: $LMT^{-1}LT^{-1} = L^2MT^{-2}$.

Prefixes

What is a prefix?

Prefixes are used in conjunction with units of measure to express multiples or submultiples of the unit in question. In other words they introduce a multiplier for associated numerical values and keep the range of the numerical value in bounds. The prefixes defined by the SI are decimal multipliers and cover the range from 10^{24} (*yotta*) to 10^{-24} (*yocto*) as can be seen in the Table 2: Decimal prefixes according to the SI below ([SP330], section 3.1, page 29).

Factor	Name	Symbol	Factor	Name	Symbol
10^{1}	deka	da	10^{-1}	deci	d
10^{2}	hecto	h	10^{-2}	centi	c
10^{3}	kilo	k	10^{-3}	milli	m
10^{6}	mega	M	10^{-6}	micro	µ
10^{9}	giga	G	10^{-9}	nano	n
10^{12}	tera	T	10^{-12}	pico	p
10^{15}	peta	P	10^{-15}	femto	f
10^{18}	exa	E	10^{-18}	atto	a
10^{21}	zetta	Z	10^{-21}	zepto	z
10^{24}	yotta	Y	10^{-24}	yocto	y

Table 2: Decimal prefixes according to the SI

Also in use are binary prefixes, especially in the electronics and computer industries. These prefixes do not contain submultiples (multipliers smaller than 1). Consult Table 3: Binary prefixes according to the IEC below for the prefixes as defined by ISO/IEC 80000-13 (taken from ([VIM], section 1.17, page 11).

Factor	Prefix Name	Prefix Symbol
$(2^{10})^{1}$	kibi	Ki
$(2^{10})^{2}$	mebi	Mi
$(2^{10})^{3}$	gibi	Gi
$(2^{10})^{4}$	tebi	Ti
$(2^{10})^{5}$	pebi	Pi
$(2^{10})^{6}$	exbi	Ei
$(2^{10})^{7}$	zebi	Zi
$(2^{10})^{8}$	yobi	Yi

Table 3: Binary prefixes according to the IEC

Combining prefixes

“Compound prefix symbols, that is, prefix symbols formed by the juxtaposition of two or more prefix symbols, are not permitted. This rule also applies to compound prefix names" ([SP330], section 3.1, page 30) - thus there is no microkilogram or µkg. Instead the related multipliers of the prefixes shall be combined to form a new prefix, `milli' in this example. UnitsML does not allow you to define a prefix consisting of multiple base-exponent pairs, or refer to more than one prefix from **<Unit>**s.

The **<Prefix>** element

The numerical multiplier is determined by the **prefixBase** and **prefixPower** attributes of the **<Prefix>** element. The name and symbol can be internationalized in the **<PrefixName>** and **<PrefixSymbol>** elements, which carry an **xml:lang** attribute for that purpose. Furthermore the **<PrefixSymbol>** element can contain either unstructured text content or any markup language, as determined by the (required) **type** attribute.

Finally prefixes are addressable and thus carry an **xml:id** attribute.

```
<Prefix prefixBase="10" prefixPower="-6" xml:id="pref_42">
   <PrefixName xml:lang="en">micro</PrefixName>
   <PrefixName xml:lang="de">mikro</PrefixName>
   <PrefixSymbol type="ASCII">u</PrefixSymbol>
   <PrefixSymbol type="unicode">µ</PrefixSymbol>
   <PrefixSymbol type="LaTeX">{\hbox{\textmu}}</PrefixSymbol>
   <PrefixSymbol type="HTML">&mu;</PrefixSymbol>
</Prefix>
```

Listing 10: Sample <Prefix> element

The example provided in Listing 10: Sample <Prefix> element above defines the `micro' prefix, i.e., a divisor of a million (base 10, power -6). The example also demonstrates the use of the **xml:lang** attribute on the **<PrefixName>** element to distinguish between the spelling of the prefix in english and in german. Also note the different **<PrefixSymbol>** elements present: The user of this information can select the type of prefix symbol that best fits their environment.

Other appearances of prefixes in UnitsML

Prefixes appear in UnitsML also in the unit's **<RootUnits>** element. On its prefix attribute a predefined list of prefixes is being used to construct derived units, e.g., square millimeter per second . The values used in this attribute correspond to the unit symbols of both Table 2 as well as Table 3, with the exception of µ. Instead of µ, 'u' is being used to accommodate non-unicode environments.

Methods of using UnitsML with other schemata

UnitsML on its own is useful, but its primary design goal was for it to be used in conjunction with other markup languages. This chapter thus discusses different ways of using UnitsML with other XML languages. The techniques shown encompass a simple reference to UnitsML content from within the target language as well as the dierent ways to compose W3C-XML Schema based languages as described in chapter four of “XML Schema Part 1: Structures Second Edition" ([XSD1], 4 Schemas and Namespaces: Access and Composition).

Each example is presented with an introduction on how the combination works, followed by the necessary modifications to the target language schema, and a sample instance document with the strategy under discussion. Finally, the benefits and drawbacks of each approach are discussed.

As a vehicle for the discussion in this chapter, assume the target language is the following `simple' language, as defined by the W3C XML Schema document in Listing 11: The 'simple' language.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Text" type="xsd:string"/>
        <xsd:element name="Measurement" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="NumericValue" type="xsd:double"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 11: The 'simple' language

The `simple' schema describes a language consisting of a blend of text elements (**<Text>**) and measurement results (**<Measurement>**). The measurement results consist of numeric values (**<NumericValue>**), which contains a double precision floating point number. A sample instance document is provided in Listing 12: A 'simple' instance document.

The task at hand now shall be to use UnitsML to unambiguously markup the unit used for the measurement (meters in this case).

```
<?xml version="1.0" encoding="UTF-8"?>
<SimpleSchemaRoot xsi:noNamespaceSchemaLocation="SimpleSchema0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Text>The width of the room is: </Text>
  <Measurement>
    <NumericValue>3.14159</NumericValue>
  </Measurement>
  <Text>meters. The width has been obtained by ...</Text>
</SimpleSchemaRoot>
```

Listing 12: A 'simple' instance document

Reference a unique unit ID

One of the simplest methods of distinguishing a scientific unit of measure is to provide a unique identifier for the desired unit, usually based on the unit's symbol or name, e.g., `m' or `meter'. This reference can be either part of the textual content representing, for example, measurement results, or it could be maintained from an attribute of complex data containing said results. A simple method of using this approach has been shown above already (the usage of `meters' in the text following the **<Measurement>** element.

In contrast to mixing the information about the used units of measure into the text itself, it is preferable to put this information into either an attribute or an element, so it can be retrieved more easily from a well-defined position and in a single expected format.

A first thought might be to use an attribute of type **xsd:token** or **xsd:string** to distinguish the different units of measures being used.

Referencing by unit symbols or names has problems, though, especially with some symbols not being as unique as one would hope (consider nm - nanometers or nautical miles?). An alternative is to have the complex data containing the marked up value point to a portion of valid UnitsML, by using a data type of **xsd:anyURI**, which gives the possibility to either address a UnitsML element within the same document (via a fragment identifier) or even a UnitsML element from a central and/or authoritative source.

XML Schema modifications

To accomplish this, the `simple' schema (and, in general, the target language) will have to be modified to accommodate for the attribute pointing to the unit being used.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Text" type="xsd:string"/>
        <xsd:element name="Measurement" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="NumericValue" type="xsd:double"/>
            </xsd:sequence>
            <xsd:attribute name="unit" type="xsd:anyURI"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 13: Relevant schema: referencing a unique ID

Instance document

By referencing a unique unit ID, it is possible to refer to UnitsML content from an authoritative source, e.g., an external document, the UnitsDB or output from a local database bearing UnitsML markup. The instance document below illustrates this method. Note that the fragment identifier at the unit URL reference most likely is not the symbol of the unit, as fragment identifiers need to be unique in the whole document (and unit in general symbols are not unique).

```
<?xml version="1.0" encoding="UTF-8"?>
<SimpleSchemaRoot xsi:noNamespaceSchemaLocation="SimpleSchema1.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Text>The width of the room is: </Text>
  <Measurement unit="http://authorative.source/Units#u1">
    <NumericValue>3.14159</NumericValue>
  </Measurement>
  <Text>. The width has been obtained by ...</Text>
</SimpleSchemaRoot>
```

Listing 14: Instance document: referencing a unique ID

Note that `meters' has been removed from the **<Text>** following the **<Measurement>** . Instead this information (the name of the unit) can be retrieved from the UnitsML markup by following the provided reference, depending on the natural language being used (e.g., English).

Discussion

Adding a pointer to the unit has the least impact on your language and its schema. By adding a single attribute, the units of measure being used can be marked up. If the URI actually resolves to some UnitsML content, it can be fetched and inspected for required information, say, pulling the unit names out of

a central database in a given language, or get information on how to convert between different units.

A bonus of this approach is that it encourages use of a dictionary of units. This keeps the information overhead slim at places which are expected to carry a lot of data. The dictionary may even reside in the same document (as realized by Listing 20: Relevant schema: importing UnitsML (alternative) on page 42).

There are some practical issues with this approach though. The W3C XML Schema validation only can ensure that the type of the attribute matches a certain pattern. If a document is valid that does not mean that the URIs resolve at all, or to some meaningful content. If the URL referenced is on a foreign server, the data might not be accessible at all places that the document is being viewed. The data might not be under the control of the document author.

Even if the references are only local URIs (e.g., using the **`file://`** `protocol'), they have to be deployed into the expected location, the XML processor in use must implement the **`xml:id`** specification ([XMLID]) or implement support for the **`xsd:ID`** data type to be able to pull the information with a good performance.

Lastly another complaint might be that the unit of measure being used isn't `seen'. Instead of a comprehensible unit reference most likely an opaque identifier will be used. Depending on the scope of the target language this might be a drawback.

In conclusion this is a low-impact way to reference units unambiguously. Its drawbacks have been sketched out above, but the ease with which it can be included speaks for itself. Also with some approaches (e.g., referencing a UnitsML instance document that makes up the `local units database' containing only the (few) used units so that meaningful **`xml:id`** attributes can be used, with a well-defined system of distribution and/or deployment) most of these problems can be avoided.

Refer to the UnitsML schema

One important reason for using XML schemata is being able to compose them in different ways ([XSD1], 4 Schemas and Namespaces: Access and Composition). In this and the following sections the different mechanisms for composing schemata will be discussed in light of using UnitsML from the `simple' schema. All four of these usage scenarios are tightly coupled with the use of namespaces in XML ([XSD1], chapter 4; [XMLNAMES11]).

By referring to the UnitsML schema the main work of pulling in the schema is not done from the target language schema, but instead in the instance documents of the target language. From the point of view of the target language schema, the foreign (in this case UnitsML) content just consists of a black box, which can be allowed at the fitting places.

XML Schema modifications

The target language schema only has to be modified in minor ways; the `burden' of the integration lies mostly within the instance documents. UnitsML elements can be used within the target language by use of the **<xsd:any>** element, as shown in Listing 15: Relevant schema: referring to UnitsML or by explicitly pulling in (a subset of) UnitsML markup. The use of the XML namespaces here helps resolve the source schema for the used elements. In essence elements (just as the other declarations of the included schema) can be used within the target language schema while they are keeping the namespace as declared as **targetNamespace** in the included XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://unitsml.nist.gov/simple"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://unitsml.nist.gov/simple"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema modified by
      referring to the schema within an XML instance document.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Text" type="xsd:string"/>
        <xsd:element name="Measurement" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="NumericValue" type="xsd:double"/>
              <xsd:any processContents="strict" minOccurs="0"
        namespace="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 15: Relevant schema: referring to UnitsML

Note that in the example in listing 3.5, the allowable namespace for the **<xsd:any>** has been set to that of UnitsML. Additionally the **processContents** attribute has been set to “strict", meaning that at this place in instance documents only well-formed, valid UnitsML-XML is allowed. Furthermore the `simple' language now exists in its own namespace.

Instance document

To assist resolving the different source schemata, the **xsi:schemaLocation** and **xsi:noNamespaceSchemaLocation** attributes will be used (cf. [XSD1], section 2.6.3).

The instance document shown in Listing 16: Sample instance document: referring to UnitsML utilizes two namespaces, “simple” and “unitsml”, to distinguish the portions of the document corresponding to the appropriate schema,

SimpleSchema2.xsd or **UnitsML-1.0.xsd**, respectively, as dictated by the **xsi:schemaLocation** attribute.

UnitsML content marking up the units of measure being used can just be dropped into place, while remaining optional (in this example).

```
<?xml version="1.0" encoding="UTF-8"?>
<simple:SimpleSchemaRoot
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:simple="http://unitsml.nist.gov/simple"
    xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
    xsi:schemaLocation="http://unitsml.nist.gov/simple
      SimpleSchema2.xsd
      urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0
     UnitsML-1.0.xsd">
  <simple:Text>The width of the room is: </simple:Text>
  <simple:Measurement>
    <simple:NumericValue>3.14159</simple:NumericValue>
    <unitsml:Unit xml:id="u42">
      <unitsml:UnitName xml:lang="en-US">meter</unitsml:UnitName>
      <unitsml:UnitSymbol type="ASCII">m</unitsml:UnitSymbol>
    </unitsml:Unit>
  </simple:Measurement>
  <simple:Text>. The width has been obtained by ...</simple:Text>
</simple:SimpleSchemaRoot>
```

Listing 16: Sample instance document: referring to UnitsML

In this case, the first referenced schema location is the host schema and the second the UnitsML schema. In the same way, we could reference additional schemata.

Discussion

Again this is a low impact way of allowing UnitsML content to be added to the target language. By designating places where UnitsML-namespaced content is allowed the units of measure being used can easily be marked up without really changing the target language. There is a strict separation of concern between the target language markup and the UnitsML markup, which, aside of being combined into a single document, do not interact with each other.

The drawbacks from the previous approach are mostly removed with this solution: As there are no references to data outside of the instance document, the data is independent of server availability or resolving URLs. Deployment is easy – as the data to be deployed already is present in the instance document.

The main drawback of this method is the potential duplication of information – multiple uses of the same unit have to carry the same element more than once – which can be considered bad practice. Additionally, this imposes a problem on the **xml:id** attribute of the **<Unit>** element, which has to be globally unique. Even though the same unit of measure might be used in different places in the instance document, the **xml:id** s must not be the same. Thus it is not possible to use the same **xml:id** in different places of the instance document.

Finally the usage of the **<xsd:any>** element does not allow specifying expected or required information (for example the need to provide the names, symbols and definitions of used units, or the requirement to mark up the units' quantities and dimensions). Given that all the elements in the UnitsML schema are global, any element could be inserted in the marked place, e.g., **<Unit>** or **<UnitName>** .

Combination of referring to the UnitsML schema with referencing units by ID

The previous two approaches can be combined into one which offers a low impact solution of including UnitsML into the target language. For that, elements that contain data that should be related to units of measure (or quantities etc.) need to be equipped with an **xsd:anyURI** attribute to point to the entity in question, and a section of the instance document needs to be devoted to describing units and its related concepts with UnitsML.

XML Schema modifications

As this approach is basically a combination of the previous two approaches, the modifications to the schema are similar: An attribute has been added to the **<Measurement>** element and a section in the instance document has been reserved to contain UnitsML markup. This is again accomplished by the use of the **<xsd:any>** element, restricting the namespace of the elements at this spot to that of UnitsML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="Text" type="xsd:string"/>
          <xsd:element name="Measurement" minOccurs="0">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="NumericValue" type="xsd:double"/>
              </xsd:sequence>
              <xsd:attribute name="unit" type="xsd:anyURI"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
        <xsd:any minOccurs="0" maxOccurs="unbounded"
          namespace="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
          processContents="strict"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 17: Relevant schema: referencing a local units database

Instance document

The instance document in Listing 18: Sample instance document: referencing a local units database shows an instance of such an approach. Now there exists a single location in the instance document where the units of measure are being described, the elements which contain data measured in some unit merely point to the unit markup at this location.

```
<?xml version="1.0" encoding="UTF-8"?>
<SimpleSchemaRoot
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
    xsi:schemaLocation="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0
      unitsmlSchema-1.0.xsd"
      xsi:noNamespaceSchemaLocation="SimpleSchema6.xsd">
  <Text>The width of the room is: </Text>
  <Measurement unit="#u42">
    <NumericValue>3.14159</NumericValue>
  </Measurement>
  <Text>. The width has been obtained by ...</Text>
  <unitsml:Unit xml:id="u42">
    <unitsml:UnitName xml:lang="en-US">meter</unitsml:UnitName>
    <unitsml:UnitSymbol type="ASCII">m</unitsml:UnitSymbol>
  </unitsml:Unit>
</SimpleSchemaRoot>
```

Listing 18: Sample instance document: referencing a local units database

Discussion

This approach combines the benefits of the previous two approaches, while eliminating most of the drawbacks: Adding the pointer to existing elements is a lean modification, and the single location of units markup encourages following the `Don't Repeat Yourself' approach common in data modeling, resolving the drawback of the previous approach (as detailed in the Discussion section of Refer to the UnitsML schema).

The drawbacks of the first approach (as outlined in the Discussion section of Reference a unique unit ID) are not reproduced: with the unit markup in the instance document there is no problem regarding deployment or availability of information.

The practical issue of having to use an XML processor which supports the **`xml:id`** specification ([XMLID]) remains though. Additionally, by needing to have a section on the units of measure being used in each instance document, there might be a considerable space overhead depending on the different operation scenarios of the target language.

Finally this method does not allow restriction to a subset of UnitsML, or enforcing usage of a whole (sub)tree of UnitsML in the `units database' section, as the **`<xsd:any>`** element will make any UnitsML element acceptable at this spot.

<import> the UnitsML schema

Another way of incorporating foreign schemata into the target language consists of using the **<import>** directive. It is described in detail in section 4.2.3 of the XML Schema Structures Part ([XSD1], section 4.2.3 References to schema components across namespaces).

Importing another schema is namespace aware and keeps the original namespace on the imported declarations and definitions. In contrast to merely referring to the UnitsML schema from within an instance document, now it is possible to refer to UnitsML content also from within the target language's schema – allowing greater control over which parts of the UnitsML schema are being used in the target language.

XML Schema modifications

The `simple' schema in Listing 19: Relevant schema: importing UnitsML now was modified to pull in the UnitsML schema. This is accomplished with the use of the **<xsd:import>** element. In contrast to the previous example, here only the **<Unit>** element is being used in the target language, accomplished by referring to its definition within the UnitsML schema as follows: **<xsd:element ref="unitsml:Unit"/>** .

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:simple="http://unitsml.nist.gov/simple"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
  targetNamespace="http://unitsml.nist.gov/simple"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:import
   namespace="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
    schemaLocation="UnitsML-1.0.xsd"/>
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema modified by
importing
        the UnitsML schema into the Simple schema.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Text" type="xsd:string"/>
        <xsd:element name="Measurement" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="NumericValue" type="xsd:double"/>
              <xsd:element ref="unitsml:Unit"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 19: Relevant schema: importing UnitsML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:simple="http://unitsml.nist.gov/simple"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
targetNamespace="http://unitsml.nist.gov/simple"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:import namespace="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-
1.0"
    schemaLocation="UnitsML-1.0.xsd"/>
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema modified by
importing
        the UnitsML schema into the Simple schema.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="Text" type="xsd:string"/>
          <xsd:element name="Measurement" minOccurs="0">
            <xsd:complexType>
              <xsd:sequence maxOccurs="unbounded">
                <xsd:element name="NumericValue" type="xsd:double"/>
              </xsd:sequence>
              <xsd:attribute name="unit" type="xsd:anyURI" use="required"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
        <xsd:element ref="unitsml:UnitSet"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 20: Relevant schema: importing UnitsML (alternative)

This approach places the UnitsML element **<UnitSet>** as last in the language, allowing a `local units database'.

Instance document

Using this option of composition, a `simple' data file could look like the example shown in Listing 21: Sample instance document: importing UnitsML. It shows that the `simple' namespace (**xmlns:simple**) is different than the UnitsML namespace (**xmlns:unitsml**) and that the units part of the document is described completely in UnitsML. Note that the UnitsML namespace does not need to be resolved to its defining schema as the **<xsd:import>** already resolves the schema location.

```
<?xml version="1.0" encoding="UTF-8"?>
<simple:SimpleSchemaRoot
  xsi:schemaLocation="http://unitsml.nist.gov/simple SimpleSchema4.xsd"
  xmlns:simple="http://unitsml.nist.gov/simple"
  xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <simple:Text>The width of the room is: </simple:Text>
  <simple:Measurement>
    <simple:NumericValue>3.14159</simple:NumericValue>
    <unitsml:Unit xml:id="u42">
      <unitsml:UnitName xml:lang="en-US">meter</unitsml:UnitName>
      <unitsml:UnitSymbol type="ASCII">m</unitsml:UnitSymbol>
    </unitsml:Unit>
  </simple:Measurement>
  <simple:Text>. The width has been obtained by ...</simple:Text>
</simple:SimpleSchemaRoot>
```

Listing 21: Sample instance document: importing UnitsML

An alternative to using the explicit namespace prefix (“**`simple:`**”) for the `simple' language would be to globally declare the default namespace to be that of the target language, in this case by replacing the **`xmlns:simple`** attribute on the **`<SimpleSchemaRoot>`** by solely **`xmlns`** and dropping all the **`simple:`** prefixes within the instance document.

```
<?xml version="1.0" encoding="UTF-8"?>
<simple:SimpleSchemaRoot
  xsi:schemaLocation="http://unitsml.nist.gov/simple SimpleSchema7.xsd"
  xmlns:simple="http://unitsml.nist.gov/simple"
  xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <simple:Text>The width of the room is: </simple:Text>
  <simple:Measurement unit="#u42">
    <simple:NumericValue>3.14159</simple:NumericValue>
  </simple:Measurement>
  <simple:Text>. The width has been obtained by ...</simple:Text>
  <unitsml:UnitSet>
    <unitsml:Unit xml:id="u42">
      <unitsml:UnitName xml:lang="en-US">meter</unitsml:UnitName>
      <unitsml:UnitSymbol type="ASCII">m</unitsml:UnitSymbol>
    </unitsml:Unit>
  </unitsml:UnitSet>
</simple:SimpleSchemaRoot>
```

Listing 22: Sample instance document: importing UnitsML (alternative)

In contrast to the instance document in Listing 21: Sample instance document: importing UnitsML, the UnitsML portion can be placed differently as outlined in Listing 20: Relevant schema: importing UnitsML (alternative). A resulting instance document could look like the one in Listing 22: Sample instance document: importing UnitsML (alternative).

Discussion

This approach results in benefits and drawbacks similar to the ones discussed in Refer to the UnitsML schema on page 36 with an added benefit: by bringing the UnitsML schema into the target language schema, all of the UnitsML declarations

are ready to be deployed in the target language schema, giving fine-grained control over which elements are allowed at which places.

Just as the discussed solution in Combination of referring to the UnitsML schema with referencing units by ID on page 39 combines the benefits of pointing to unit definitions via an **`xsd:anyURI`** attribute with keeping the `database' of units locally, the alternative approach outlined in Listing 20: Relevant schema: importing UnitsML (alternative) for the schema and in its instance document in Listing 22: Sample instance document: importing UnitsML (alternative) combines these for the case of importing the UnitsML schema into the target language. This should demonstrate that a wise choice of selection of elements from UnitsML together with a well-informed choice of placing the usage of these definitions can be decisive on the resulting set of positive and negative influences on the target language.

`<include>` the UnitsML schema

Another option of incorporating foreign schemata into the target language consists of using the **`<include>`** directive. It is described in detail in section 4.2.1 of the XML Schema Structures Part ([XSD1], section 4.2.1 Assembling a schema for a single target namespace from multiple schema definition documents).

In contrast to using the previous approaches, this approach needs modifications of the UnitsML schema, particularly as including another schema requires the included schema to either have no target namespace, or to have the same namespace as the including schema. To accomplish the equality of the namespaces for this (and the following) example, the stylesheet in Listing 23: XSLT stylesheet to prepare the UnitsML schema for inclusion and redefinition has been used to completely remove namespaces from the UnitsML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  stylesheet to strip namespaces from the OASIS UnitsML schema which
  has a home under:
   * OASIS: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=unitsml
   * NIST: http://unitsml.nist.gov

-->
<xsl:stylesheet version="1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
  xmlns:docco="http://www.ayaken.net/xsd2docco/2010/1"
  exclude-result-prefixes="xsl unitsml">
  <xsl:output method="xml" encoding="UTF-8" cdata-section-
elements="docco:example" />

  <xsl:template match="/xsd:schema">
    <xsd:schema>
      <!-- attributes must come first: copy everything except the
targetNamespace attribute. -->
      <!-- ..also overwrite element- and attributeFormDefault to make sure they
are 'unqualified' -->
      <xsl:for-each select="@*">
        <xsl:choose>
          <xsl:when test="local-name(.) = 'elementFormDefault'">
            <xsl:attribute name="elementFormDefault">unqualified</xsl:attribute>
          </xsl:when>
          <xsl:when test="local-name(.) = 'attributeFormDefault'">
            <xsl:attribute
name="attributeFormDefault">unqualified</xsl:attribute>
          </xsl:when>
          <xsl:when test="local-name(.) = 'targetNamespace'">
            <!-- do nothing -->
          </xsl:when>
          <xsl:otherwise>
             <xsl:copy />
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
      <!-- now for namespaces: copy all except the default namespace
declaration. -->
      <xsl:for-each select="namespace::*">
        <xsl:if test="local-name(.) != ''">
          <xsl:copy />
        </xsl:if>
      </xsl:for-each>
      <!-- now copy all the rest -->
      <xsl:apply-templates select="node()" />
    </xsd:schema>
  </xsl:template>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Listing 23: XSLT stylesheet to prepare the UnitsML schema for inclusion and redefinition

With this stylesheet, a new schema file **`UnitsML-1.0-noNamespace.xsd`** has been created which is identical to the original schema save for removing the target namespace declarations

XML Schema modifications

The `simple' schema now pulls in the UnitsML schema via use of the **<xsd:include>** element as shown in Listing 24: Relevant schema: including UnitsML. This makes all of the definitions from the UnitsML schema available for use within the `simple' schema. The **<Unit>** element is added to the measurement element. Note that it does not carry a namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:include schemaLocation="UnitsML-1.0-NoNamespace.xsd"/>
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema modified by
including
        the UnitsML schema into the Simple schema.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Text" type="xsd:string"/>
        <xsd:element name="Measurement" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="NumericValue" type="xsd:double"/>
              <xsd:element ref="Unit"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 24: Relevant schema: including UnitsML

Instance document

The XML document in Listing 25: Sample instance document: including UnitsML is an example of an instance document with this option. Notable in contrast to the previous two approaches is the lack of the use of namespaces (although, of course, namespaces can still be used when including other schemata).

```
<?xml version="1.0" encoding="UTF-8"?>
<SimpleSchemaRoot xsi:noNamespaceSchemaLocation="SimpleSchema3.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Text>The width of the room is: </Text>
  <Measurement>
    <NumericValue>3.14159</NumericValue>
    <Unit xml:id="u42">
      <UnitName xml:lang="en-US">meter</UnitName>
      <UnitSymbol type="ASCII">m</UnitSymbol>
    </Unit>
  </Measurement>
  <Text>. The width has been obtained by ...</Text>
</SimpleSchemaRoot>
```

Listing 25: Sample instance document: including UnitsML

Discussion

In addition to the benefits and drawbacks discussed in previous sections, including the UnitsML schema has the following two traits: First, it requires editing the UnitsML schema so that the **`targetNamespace`** attribute of the standard UnitsML does not stand in the way of the target language. This process can be automated through a stylesheet like that shown in Listing 23: XSLT stylesheet to prepare the UnitsML schema for inclusion and redefinition. The benefit that is achieved through accepting this drawback though is that the target language and UnitsML no longer exist in different namespaces, thus keeping up a uniform appearance to its users. On the other hand, by removing the UnitsML namespace or changing the UnitsML namespace to match the target language, the possibility for name clashes arises.

`<redefine>` the elements of UnitsML

The **`redefine`** method is related to the **`include`** method in that the parts of the redefined schema must be in the same namespace as the host schema, whether declared or not. Thus, the version of the UnitsML schema to be redefined must have no declared namespace. The **`<redefine>`** directive is describe in detail in section 4.2.2 of the XML Schema Structures Part ([XSD1], section 4.2.2 Including modified component definitions).

For this usage scenario the same result document **`UnitsML-1.0-noNamespace.xsd`** created by the XSLT stylesheet in Listing 23: XSLT stylesheet to prepare the UnitsML schema for inclusion and redefinition has been used.

Whereas the **`include`** method does not allow changes to be made to the included schema, the **`redefine`** method allows simple and complex types, as well as element and attribute groups, to be modified by extension or restriction.

XML Schema modifications

The schema in Listing 26: Relevant schema: redefining UnitsML first redefines the **<Unit>** element from UnitsML to include a text element **<MyText>** . This modified **<Unit>** element then is used in the `simple' schema itself.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:redefine schemaLocation="UnitsML-1.0-noNamespace.xsd">
    <xsd:complexType name="UnitType">
      <xsd:complexContent>
        <xsd:extension base="UnitType">
          <xsd:sequence>
            <xsd:element name="MyText" minOccurs="0" type="xsd:string"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:redefine>
  <xsd:element name="SimpleSchemaRoot">
    <xsd:annotation>
      <xsd:documentation>Root element for simple test schema modified by
        including the UnitsML schema into the Simple schema redefining the Unit
        element by extension.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Text" type="xsd:string" />
        <xsd:element name="Measurement" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="NumericValue" type="xsd:double" />
              <xsd:element ref="Unit" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 26: Relevant schema: redefining UnitsML

Instance document

The instance document in Listing 27: Sample instance document: redefining UnitsML is identical to that for the **include** method, except for the additional element **MyText** being used.

```
<?xml version="1.0" encoding="UTF-8"?>
<SimpleSchemaRoot xsi:noNamespaceSchemaLocation="SimpleSchema5.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Text>The width of the room is: </Text>
  <Measurement>
    <NumericValue>3.14159</NumericValue>
    <Unit xml:id="u42">
      <UnitName xml:lang="en-US">meter</UnitName>
      <UnitSymbol type="ASCII">m</UnitSymbol>
      <MyText>This is sample text in an added element.</MyText>
    </Unit>
  </Measurement>
  <Text>. The width has been obtained by ...</Text>
</SimpleSchemaRoot>
```

Listing 27: Sample instance document: redefining UnitsML

Discussion

The final improvement achieved by this solution consists of being able to extend the UnitsML schema. This cannot be achieved without dropping its namespace though.

Now it is possible to create another shell around UnitsML which wraps vanilla UnitsML and extends it while keeping the original UnitsML namespace. This is strongly discouraged though, as the mechanisms for locating language schemata is closely coupled to the namespace of the language in question. In other words, when creating a different language, it should also exist in a different namespace.

Summary

In this chapter different approaches for incorporating UnitsML into the target language have been demonstrated, each with its own specific set of benefits and drawbacks. Please refer to the discussion section of each approach to evaluate in detail which solution best fits your use-case.

Sparing the details, the different approaches' pros and cons and the impact on the target language (TL) schema can be summarized as in following table:

Approach	Benefits	Drawbacks
Reference a unit	+ Few schema modifications for TL necessary + keeps units out of TL + lean & flexible approach + can reference global authoritative sources + 'Don't Repeat Yourself' fulfilled	+ No guarantee that the URIs resolve / deployment issues + cannot use UnitsML within TL schema
Refer to UnitsML	+ Few schema modifications for TL necessary + units, etc., clearly visible in instance documents + separation of concerns between TL and UnitsML + easily deployable	+ Multiple occurrences of some element result in problems with the **`xml:id`** attribute + 'Don't Repeat Yourself' violated + cannot use UnitsML within TL schema + no strict schema validation possible (use of **`<xsd:any>`**) + burden of weaving together namespaces on each instance document
Combination of above	• Few schema modifications for TL necessary • easily deployable • units, etc., well visible in 'local units database' • flexible solution (refer to local and foreign content) • 'Don't Repeat Yourself'	• Cannot use UnitsML within TL schema • no strict schema validation possible (use of **`<xsd:any>`**) • burden of weaving together namespaces on each instance document

Approach	Benefits	Drawbacks
	fulfilled	
<import> UnitsML	• Full access to UnitsML from the TL schema – fine grained validation possible • no instance document namespace / schema location synthesis necessary • depending on placement (cf. Listing 20: Relevant schema: importing UnitsML (alternative)), flexible solution	• Extensive target language schema modifications necessary • requires well-planned integration
<include> UnitsML	• Pull UnitsML into TL namespace • full access to UnitsML from the TL schema – fine grained validation possible • no instance document namespace / schema location synthesis necessary • depending on placement (cf. Listing 20: Relevant schema: importing UnitsML (alternative)), flexible solution	• Requires modification of the UnitsML schema • extensive schema modifications necessary • requires well-planned integration
<redefine> UnitsML	• Same as **<include>** above, plus • allowing to extend UnitsML	• Same as **<include>** above, plus • redefinitions depart from OASIS UnitsML

TABLE 4: SUMMARY OF DIFFERENT UNITSML USAGE APPROACHES

Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Robert Dragoset, NIST
Martin S. Weber, NIST
Karen Olsen, NIST
Mark Carlisle, NIST
Peter Linstrom, NIST
Karen LeGrand, IEM

We also gratefully acknowledge the input of former members of the OASIS UnitsML technical committee in forming the UnitsML schema that is described herein.

Non-Normative Section

W3C XML Schema for UnitsML

Naming and Design Rules for UnitsML

Revision History

Revision	Date	Editor	Changes Made
[Rev number]	[Rev Date]	[Modified By]	[Summary of Changes]