Another possible model for a Quantity

Lowe and McDowell suggest a model of a quantity which is divided into two components; the numerical value(s), and a *representation* which identifies the coordinate system in which the Quantity is expressed and encapsulates the extra information needed to transform the Quantity to another system. A model very similar to this has been in use within the UK Starlink project for several years, and this note outlines this model. The focus is on the Starlink equivalent to the Lowe/McDowell Representation class, which I will here call a *Domain*.

The Domain class:

A Domain can be thought of a describing a set of coordinate systems which can be used to represent physically equivalent quantities. Thus wavelength, frequency, energy, *etc.* are different coordinate systems which can be used to represent position within an electro-magnetic spectrum—they all represent the same physical quantity. Again, a position on the sky can be represented in many different coordinate systems, as can a moment in time, or a flux. Thus a Domain represents a physical domain within which one or more prescribed coordinate systems can be used to describe positions.

The base Domain class has 3 main properties:

- *Name*: This is a name (possibly a UCD) which identifies the physical quantity represented by the Domain. Note, this name does not include any details of the specific coordinate system used to describe positions within the Domain (this is the job of the *System* property described below), it just identifies the physical domain described by the Domain. For instance, "WAVELENGTH" would not be appropriate for *Name.* Instead, "SPECTRUM" (or some such thing) should be used. This is because wavelength is just one of many *coordinate systems* which can be used to describe positions within a spectral *domain.* The *Name* property identifies the domain, not the coordinate system. "TELESCOPE_FOCAL_LENGTH" on the other hand *would* be suitable value for *Name.*
- *System*: This is an identifier for a specific coordinate system, selected from the list of coordinate systems supported by the Domain. This is the coordinate system in which the Quantity value is expressed. The base Domain class would probably support just a single default Cartesian coordinate system. This would be appropriate for instance for a Domain representing telescope focal length—there is only one coordinate system in which you can measure the focal length of a telescope (albeit you may express axis values within that coordinate system using different units, "m", "mm", *etc).*
- *Unit[]*: An array of unit specifiers, one for each axis in the coordinate system given by *System.*

This base Domain class could be used to describe many simple domains for which a Unit specification is all that is needed. In many cases it may be appropriate to leave the *Name* and *System* properties unset—for instance, if you wanted to describe some unspecified length you would just set the *Unit* property to "m", "lyr" or whatever, and leave the other properties unset. Thus, simple things are simple.

The main method on the base Domain class would be a method which took two Domains and produced a Mapping which transforms one into the other, taking account of any difference in coordinate system or units. This Mapping could then be used to transform the Quantity value into the new Domain.

Sub-classing Domain to describe more complex quantities:

Sub-classes of Domain could then be produced to cater for those physical domains which have more than one possible coordinate system. I will use the example of the spectral domain to illustrate.

A SpectralDomain inherits the *Name* and *Units* properties from the Domain class, and over-rides the *System* property to allow a range of spectral coordinate system to be specified. For instance, you could adopt the list specified in the FITS-WCS paper III on spectral representations within FITS: wavelength, frequency, energy, radio velocity, optical velocity, relativistic velocity, wave-number, redshift, beta-factor and wavelength-in-air. The SpectralDomain class then extends the Domain class by adding properties to hold the information needed to define these coordinate systems. This ensures that a position in any of these coordinate systems can be transformed into the corresponding position in any of the others. These extra properties would include, rest frame, observers position, date of observation, position of the source, rest frequency, *etc.*

Likewise, sub-classes of Domain could be produced to describe positions on the sky, or moments in time, or fluxes, *etc*., each extending Domain to allow a suitable range of values for *System*, and to add any extra information needed to define the Mappings between the allowed coordinate systems.

Note, a Quantity representing a wavelength (described by a SpectralDomain) could be combined with a telescope focal length (described by a simple base Domain), simply by casting the SpectralDomain into a Domain, thus throwing away all the spectral-specific properties and behaviour of the SpectralDomain. This would result in the wavelength being treated as a simple length.

How would it all work?

I will illustrate how this model would work in the context of comparing two Quantities (assuming that both Quantities represent scalars):

```
Quantity q1, q2;
if( q1.greaterThan( q2 ) ) {
   …
```

The implementation of the `greaterThan` method would include (this is a simplified implementation just to illustrate the main points):

```
public boolean greaterThan( Quantity q ){

   /* Get the domains in which the two quantities reside. */
```

```
    Domain dom1 = this.getDomain();
    Domain dom2 = q.getDomain();

    /* Get a Mapping which transforms positions in the coordinate
       system of dom2 into the coordinate system of dom1 (if
       possible). */
    Mapping map = dom2.getMapping( dom1 );

    /* Throw an exception if no Mapping can be found between the
       two Domains. */
    if( map == null ) {
        throw new IllegalArgumentException( "Cannot convert a "+
                    q.toString()+" into a "+this.toString()" );
    }

    /* Get the value of the supplied quantity and convert into the
       coordinate system of this quantity. */
    double val2 = map.transform( q.getValue() );

    /* Now we have the Quantities in the same coordinate system, do
       the comparison. */
    return ( this.getValue() > val2 );
  }
```

The bulk of the cleverness is in the `Domain.getMapping` method.

Previous experience with this model:

Starlink have been using a system like this for several years and have found it to work well within the context of transforming Quantities involved in WCS descriptions (see the AST library docs at `http://www.ast.man.ac.uk/~dsb/ast/ast.html`). I see no reason why it could not be extended to other Quantities more usually associated with observables rather than coordinates. Note:

- within the context of AST, what I have been calling a *Domain* is called a *Frame* (I think the word Domain more clearly specifies its use in describing a physical domain).
- AST has no direct equivalent to Quantity—the responsibility for associating a Domain with a collection of axis values is left up to the caller.