Engineering units is an important concept in software engineering and in any engineering branch. Create a system that enables the follwoing.

1. The system shall be based on the POSC Unit of Measure Dictionary v2.2 (http://w3.energistics.org/uom/poscUnits22.xml). A guide for the dictionary can be found here http://w3.energistics.org/uom/Energistics_Unit_of_Measure_Usage_Guide_V1.pdf http://w3.energistics.org/uom/poscUnits22.xml
   The system must specifically adhere to the definitions in this document (does not include section 3.3.3.1, 3.3.3.1.1, 3.3.3.1.2, 3.3.4, 5.3.6 and 5.3.7)
2. The system shall enable:
   a. List all unit dimensions or any type that is specified
   b. List all quantity classes*
   c. List all uom for a given quantity class*
   d. List all aliases for a given uom
   e. Convert to/from any uom within the same base unit. Expected input is one double and two string. The expected output is the result of the conversion, the uom, and the uom annotation.
   f. Support a creating sub quantity classes. The reason for this is that if someone ran 10.000 meters, and wanted to change the engineering units from meters to miles, the user should be able to specify a sub quantity class for e.g. “running” lest mm be in the list.

   *quantity class is the same as quantity type.

3. The system must support round-trip conversion, and conversion to and from any unit within the same unit of measure.
4. The conversion component should opt for high performance, low memory footprint.

Select one -ility of choice. Implement the tactics you feel appropriate. Document which -ility you chose and how this tactic fulfils the -ility.

Create the following design documents:

1. Module view (module diagram)
2. Class view for every module (class diagram)
3. At least two interactions views (interaction diagrams)
   a. One for the first call of conversion.
   b. One for a subsequent call of a previous conversion

Implement the system described above that fits the following criteria.

1. As a web service, using either REST/JSON or SOAP/XML, for this you should also opt for high availability.
2. As a component for a thick client application or in-house system with no internet connection. For thick clients, the user should have an easy way of implementing, changing, modifying units, classes, types etc.