Specifying Units of Measure in POSL and RuleML 1.0: Targeting OO jDREW Execution

Supervisor

Dr. Harold Boley

Advisor

Dr. Tara Athan

Authors

Chandan Bagai
Sunil Kumar Ravikumar
Udit Trivedi

Table of Contents

Table of Figures

1.0. Introduction

1.1. Background

Specification of measurement units and conversion between various units is very essential in the field of Science, Engineering, Manufacturing, Commerce, Medicine and Environmental Regulation [3].

Seven base dimensions:

SI base unit		
Base quantity	**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

Fig. 1 *Seven Base Dimensions*

Standardizing the units of measurement is very important for ***uniformity*** and ***universality***. A quantity is a hypothetically measurable amount of some physical-dimensions. Time, length, mass, and energy are a few examples of many physical-dimensions. Quantities are described in terms of reference quantities called units of measure. A meter is an example of a unit of measure for quantities of the length physical-dimension [5].

1.2. Scope

Our project aims at building a library for specifying Units of Measure in POSL and RuleML 1.0, by exploring Tom Gruber's pioneering work on Ontolingua and OASIS Quantities and Units of Measure Ontology Standard (QUOMOS). The knowledge base has been built and executed in OO jDREW to convert a quantity of a physical dimension from one unit to other unit and also representing derived units in terms of base units. e.g. Kilometer to Meter where we represent Kilometer in terms of Meter, since Meter is the SI unit for length dimension and all other units (e.g. foot, inch, centimeter, mile etc) are measured in terms of base unit of dimension.

Second part of the project is invertibility of transformation of units, conversion of similar kind of units using the relationalizing concepts of functional logic programming. Initial idea was to adopt or implement invertible built-ins, e.g. using constraint propagation techniques. Given the limited time, we chose to use unidirectional converters, e.g. from Fahrenheit to Celsius and vice versa.

2.0. Overview

Our project involves a hierarchy of expressivity which will be specified in a subset of First Order Logic (FOL), namely Hornlog and even Datalog. As per Tom Gruber's theory on units of measure with respect to ontologies, there is a hierarchy of concepts, commonly called taxonomy and there is a general class of dimensions; subclasses of this class are base dimensions and derived dimensions. Derived dimensions are obtained by means of mathematical operation of multiplication and division on one or more base dimension. Refer to fig.1.

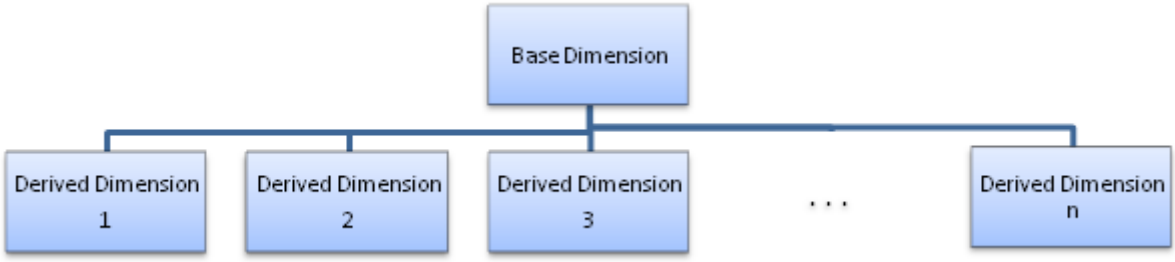


Fig. 2 *Base Dimension and Possible Derived Dimensions*

For example, the derived dimension velocity is obtained from two base dimensions; length and time. A derived dimension for the derived quantity velocity (distance divided by time) is meter per second, symbolically, m/s.

Fig. 3 shows derived units of physical dimension ***Length,*** the SI base unit of which is meter.

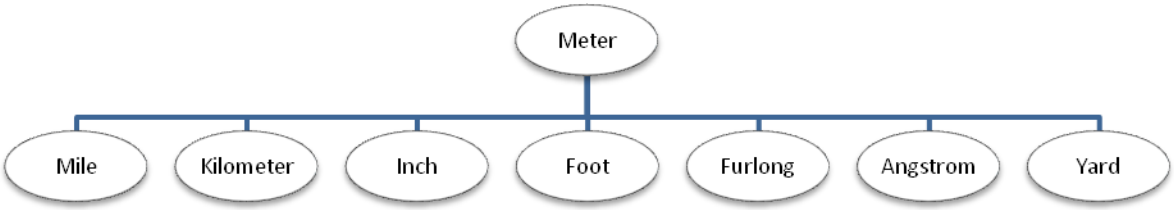


Fig. 3 *Base unit and Derived unit (For Length dimension)***.**

The SI unit for Length is Meter and we represent all other measures of length in terms of SI unit e.g. Kilometer is meter*1000 and so on.

3.0. Design

3.1. Previous Work

We researched on Tom Gruber's work on units of measure with respect to ontologies and we have used the axioms defined by Tom Gruber in our project.

Every physical quantity has a dimension and it is defined as below as per Tom Gruber.

:def (defined (quantity.dimension ?x))

The physical dimension of Area is defined as length squared,

:axiom-def (= Area

*(*LENGTH-DIMENSION *LENGTH-DIMENSION))*

while the physical dimension of Force is defined as mass times length over time squared.

:axiom-def (= FORCE-DIMENSION

(MASS-DIMENSION*

(LENGTH-DIMENSION (expt TIME-DIMENSION -2)))))*

With respect to the above mentioned axioms; Mass, Length and Time are instances of base-dimension class and Force is instance of derived-dimension class [5] .

Tom Gruber has defined the axioms for units of measure layered on the definition of **base dimensions** (i.e. refer to Fig.1, Seven base dimensions) which are used as the foundation for implementing this project.

Our researches continued with analysing the findings of OASIS Quantities and Units of Measure Ontology Standard (QUOMOS) and the same concepts are being used in our project.

3.2. Current Project

Based on the previous work, we have created a Knowledge base in OO jDREW using POSL and RuleML 1.0 and we have used the different relational languages based on the number of arguments associated with the relation. These relational languages are described below.

Binary Datalog

In Binary Datalog, relations can be used only with two arguments (e.g. two basic dimensions or constants, such as area(?square_meter, ?length), where first and second argument of rule "area" are output and input respectively.

Datalog

For n-ary relations with n>2, Binary Datalog is not sufficient for expressivity, but we can use general Datalog e.g. Acceleration can be expressed in terms of basic dimension as

```
Acceleration(?Acceleration,?Length,?Time) :-
                    multiply(?Timesquare,?Time,?Time),
                    divide(?Acceleration,?Length,? Timesquare).
```

Multiples, sub-multiples of base dimensions are obtained either by multiple quantities or divide quantities, which are "multiply dimensions" or "divide dimensions". We have used the tables mentioned below for specifying base units in terms of multiples and sub multiples of base units.

Multiples and sub- multiples	**Special symbol**	**Constant**	**Relation**	**Base Unit**	**Dimension**
Kilometer	Km	1000	M*1000	M	Length
Centimeter	Cm	100	M/100	M	Length
Inch	In	39.37	M/39.37	M	Length
Foot	Ft	12	Inch *12	Inch	Length
Mile	mi,ml,m,M	5280	Foot*5280	Foot	Length
Angstrom	ångström	10^10	M/10^10	M	Length

Yard	Yard	1.09361	M/1.09361	M	Length
Furlong	Furlong	0.00497	M/0.00497	M	Length
Milligram	Mg	10^6	Kg*10^6	Kg	Mass
Microgram	µg	10^9	Kg*10^9	Kg	Mass
Gram	G	1000	Kg*1000	Kg	Mass
Pound	Lb	2.2046	Kg/2.2046	Kg	Mass
Tonne	T	0.001	Kg/0.001	Kg	Mass
Ounce	Oz	35.273	Kg/35.273	Kg	Mass
Slug	Slug	0.06852	Kg/0.06852	Kg	Mass
Minute	Min	60	S/60	S	Time
Hour	h or hr	3600	S/3600	S	Time
Day	D	86400	S/86400	S	Time
Week	Week	604800	S/604800	S	Time
Month	Month	2592000	S/2592000	S	Time
Year	Year	31536000	S/31536000	S	Time
Area	Square meter		L*L	M	Length
Volume	Cubic meter		L*L*L	M	Length

Fig. 4 *Multiples and sub-multiples of base dimension***.**

Derived Units are obtained from the base dimensions either by multiplying or dividing the base quantities. The below table explains how we obtain derived units from the base units.

Derived quantity	Special name	Special symbol	Expression in terms of	
			Other SI units	SI base units
plane angle	Radian	Rad	1	$m \cdot m^{-1}$
solid angle	Steradian	Sr	1	$m^2 \cdot m^{-2}$
Frequency	Hertz	Hz		s^{-1}
Force	Newton	N		$m \cdot kg \cdot s^{-2}$
pressure, stress	Pascal	Pa	N/m^2	$m^{-1} \cdot kg \cdot s^{-2}$
energy, work, quantity of heat	Joule	J	$N \cdot m$	$m^2 \cdot kg \cdot s^{-2}$
power, radiant flux	Watt	W	J/s	$m^2 \cdot kg \cdot s^{-3}$
electric charge, quantity of electricity	Coulomb	C		$s \cdot A$
electric potential, potential difference, electromotive force	Volt	V	W/A	$m^2 \cdot kg \cdot s^{-3} \cdot A^{-1}$
Capacitance	Farad	F	C/V	$m^{-2} \cdot kg^{-1} \cdot s^4 \cdot A^2$
electric resistance	Ohm	Ω	V/A	$m^2 \cdot kg \cdot s^{-3} \cdot A^{-2}$
electric conductance	Siemens	S	A/V	$m^{-2} \cdot kg^{-1} \cdot s^3 \cdot A^2$

magnetic flux	Weber	Wb	V · s	m2 · kg · s−2 · A−1
magnetic flux density	Tesla	T	Wb/m2	kg · s−2 · A−1
Inductance	henry	H	Wb/A	m2 · kg · s−2 · A−2
Celsius temperature	degree Celsius	°C		K
luminous flux	Lumen	Lm	cd · sr	Cd
illuminance	Lux	Lx	lm/m2	m−2 · cd
activity (of a radionuclide)	Becquerel	Bq		s−1
absorbed dose, specific energy (imparted), kerma	Gray	Gy	J/kg	m2 · s−2
dose equivalent, et al.	Sievert	Sv	J/kg	m2 · s−2
catalytic activity	Katal	Kat		s−1 · mol

Fig. 5 *Derived units***.**

The tables mentioned in fig.4 and fig.5 is used for implementing the code in POSL and Rule ML 1.0 and the concept of relationalizing the functions is used to build invertible built-ins.

4.0. Methodology

We have used an Incremental Development Model in our project. Refer to fig. 6.

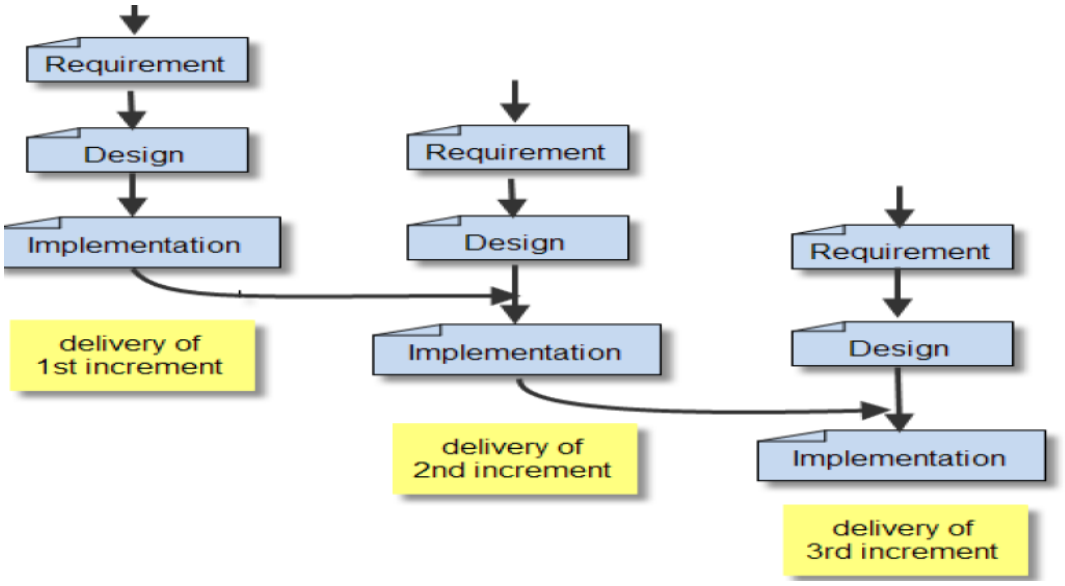


Fig. 6 *Three Development Increments*

We have accomplished our goal by implementing the following tasks:-

Increment 1: During the first phase, we focused on relational facts. We have defined the facts which associate base dimension and their respective base units. For example, `basedimension(length,meter,m)` fact defines that meter is the unit for base dimension length.

Moreover, facts required for derived dimensions were also developed. For example, the below fact is used to convert meter into kilometers where 1000 is the numeric constant which is multiplied to the base dimension to get the sub-multiple of base dimension.

`lengthmeasure(1000:Real,kilometer,meter).`

Increment 2: In the second increment, we concentrated on facts and rules to derive quantities measured in units from those base units. For example, Meter to Kilometer; where kilometer is a derived unit of length and meter is the base unit of length. Later we continued with facts and rules for other derived dimension, like Force, Density, and Acceleration.

```
Density(?Density:Real,?Mass:Real,?Length:Real,?Unit) :-
     derivedunit(density, ?Unit),
     Volume(?Volume:Real,?Length:Real,?UnitVol),
     divide(?Density:Real,?Mass:Real,?Volume:Real).
```

Increment 3: The third and last phase focused on implementing invertibility of transformations between units using relationalizing of the functions e.g. Conversion of Celsius to Fahrenheit and vice versa.

```
fahrenheit2celsius(?Celsius,?Fahrenheit:Real) :-
     subtract(?Value:Real,?Fahrenheit:Real,32:Real),
     multiply(?Value1:Real,?Value:Real,5:Real),
     divide(?Celsius:Real,?Value1,9:Real).
```

We have referred to the literature work of Tom Gruber [5].

4.1. *Examples*

The OO jDREW engine is used to build the library for specifying units of measure in POSL and RuleML. Below examples explain how to interpret the output obtained from the query engine of OO jDREW.

Example-1 : Force

Syntax:-Force(?Force:Real,?Mass:Real,?Length:Real,?Time:Real, ?Unit) :-

derivedunit(force, ?Unit),

Acceleration(?Acceleration:Real,?Length:Real,?Time:Real,?UnitAcc),

multiply(?Force:Real,?Mass:Real,?Acceleration:Real).

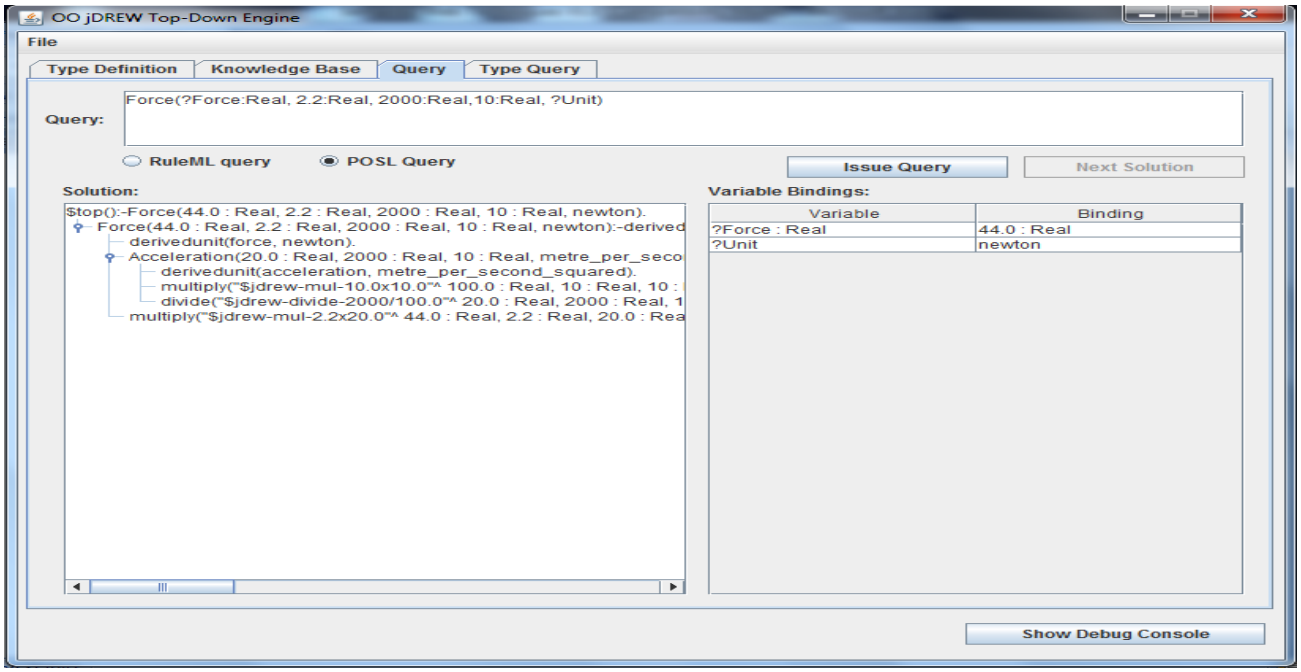
Input:- Mass - 2.2(kilogram)

Length - 2000(meter)

Time - 10(second)

Output:-Force - 44.0

Unit - newton



Example-2 : From Meter to Kilometer

Syntax:- meter2kilometer(?Kilometer:Real,?Meter:Real) :-

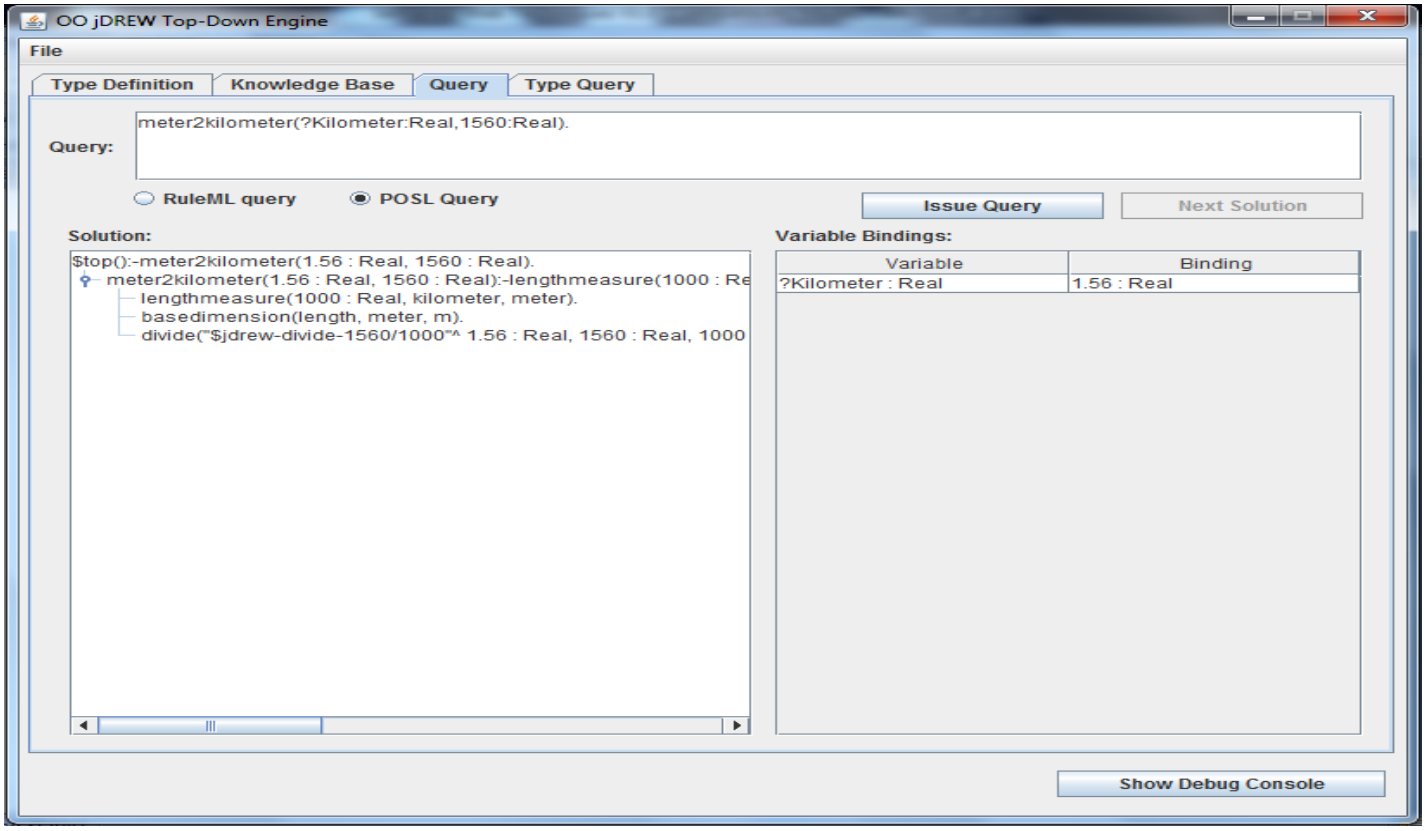
lengthmeasure(?Value:Real,kilometer,?Unit),

basedimension(?Dimension,?Unit,m),

divide(?Kilometer:Real,?Meter:Real,?Value:Real).

Input:- Meter-1560

Output:-Kilometer-1.56



Example-3 : Celsius to Fahrenheit

Syntax:- celsius2fahrenheit(?Fahrenheit,?Celsius:Real) :-

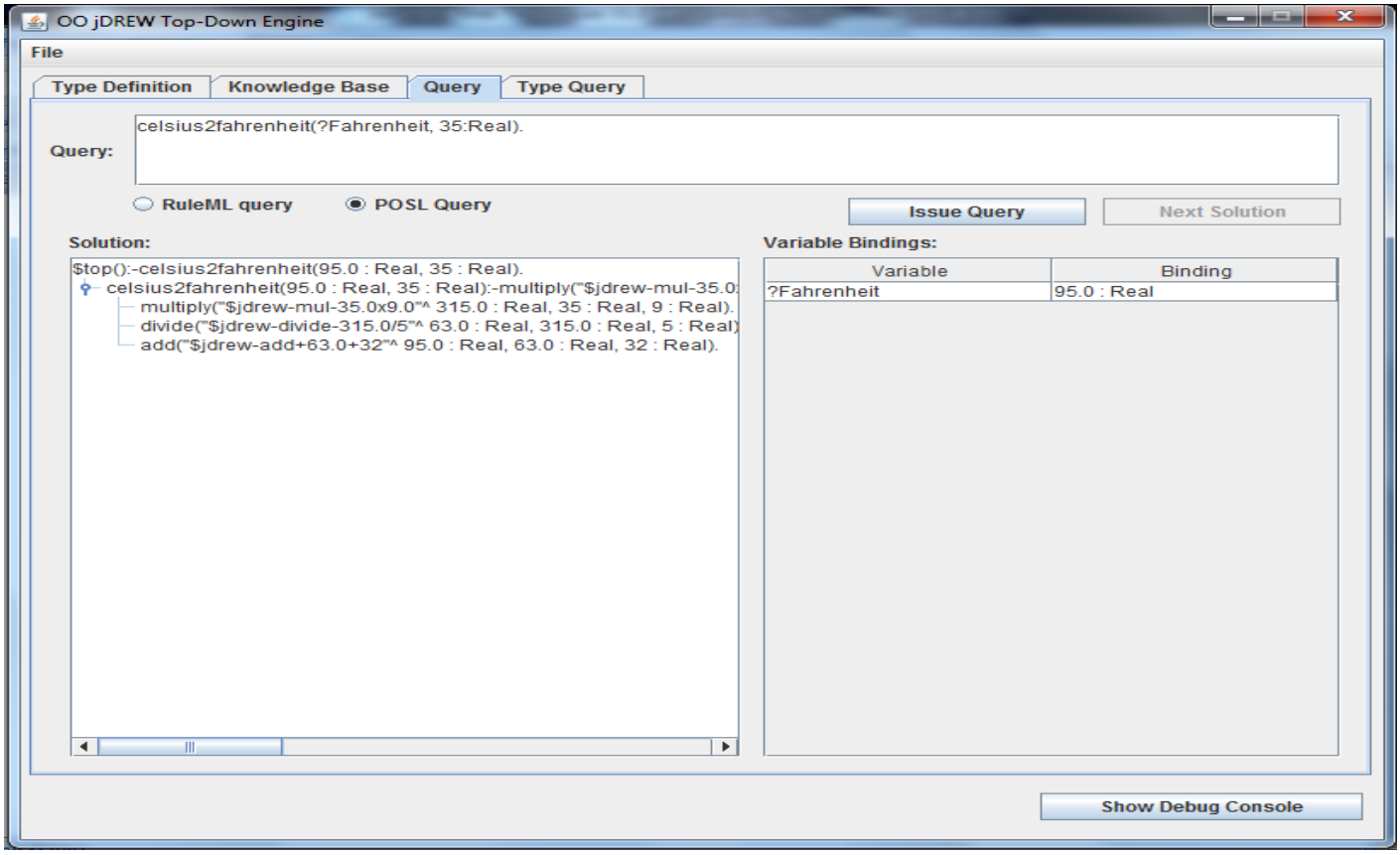
multiply(?Value:Real,?Celsius:Real,9:Real),

divide(?Value1:Real,?Value:Real,5:Real),

add(?Fahrenheit:Real,?Value1,32:Real).

Input:- Celsius - 35

Output:- Force - 95.0



5.0. Restrictions

5.1. Required Files

With the current version of OO jDREW we don't have type casting so we implemented a built-in which will convert any input argument supplied into String. We required this function for using it along with the already present built-in function String Concatenation for time based dimensions e.g. "1hour 20minute 30second" integers(i.e.1,20,30) and string(i.e. hour, minute, second) can't be concatenated in the current version of OOjDREW, we should typecast integer to string in order to concatenate; to overcome this issue we have developed "InputToString" built-in (which is defined in the appendix).

6.0. Conclusions

- Specifying units of measure and conversion between units have vast application in the fields of Science, Engineering, Manufacturing, Commerce, Medicine and Environmental Regulation.
- Various geographical regions have their own measurement unit; it becomes easier if we specify the units of measure in terms of base dimensions.
- Moreover, OO jDREW supports many pre defined built-ins, e.g. for addition, multiplication, division, mod, date and time. Further, OO jDREW is coded in Java and is an open source engine; therefore, we can also create and add our own built-ins as need arise; e.g. "InputToString" built-in explained in the appendix.

7.0. Future Work

- For most of the conversion, there are fixed multiplicative factors associated with each unit. But, the conversions between currency values of different countries are different. Based on the economical trading of the countries, currency conversion rate change frequently.
- Future work could study a web service for declarative currency conversion based on existing convertors. This web service could pull the dynamic data from a website which has exchange rates for a particular day (For example, www.x-rates.com). There will be facts in OO jDREW which will correspond to current exchange rates. Rules could be standardized using US dollar as base currency.

8.0. References

1. POSL- An Integrated *Positional-Slotted Language for Semantic Web Knowledge*

http://ruleml.org/submission/ruleml-shortation.html

2. Units of Measure

http://physics.nist.gov/Pubs/SP811/sec04.html

3. Frank Olken: Ontology of Measurement Units and Dimensions:

http://ontolog.cim3.net/file/work/OntologySummit2009/OntologySummit2009_Symposium_20090406-07/units-ontology-talk-v01--FrankOlken_20090406.pdf

4. NIST UnitsML:

http://unitsml.nist.gov/Presentations/UnitsML-SCC20.ppt

5. Ontolingua Quantities and Units:

http://www-ksl.stanford.edu/htw/dme/thermal-kb-tour/physical-quantities.html

http://www-ksl.stanford.edu/htw/dme/thermal-kb-tour/standard-units.html

6. Functional-Logic Programming

http://www.cs.unb.ca/~boley/FLP/notes.html

9.0. Appendix

9.1. POSL Code

```
basedimension(length,meter,m).
basedimension(mass,kilogram,kg).
basedimension(time,second,s).
basedimension(electriccurrent,ampere,A).
basedimension(temperature,kelvin,K).
basedimension(amountofsubstance,mole,mol).
basedimension(luminousIntensity,candela,cd).

lengthmeasure(10:Real,angstrom,meter).
lengthmeasure(1000000:Real,micrometer,meter).
lengthmeasure(1000:Real,millimeter,meter).
lengthmeasure(100:Real,centimeter,meter).
lengthmeasure(39.37 :Real,inch,meter).
Lengthmeasure(10:Real,decimeter,meter).
exponent(10:Real, angstrom).
```

```
massmeasure(1000000000:Real,microgram,kilogram).
massmeasure(1000:Real,gram,kilogram).
massmeasure(1000000:Real,milligram,kilogram).

timemeasure(60:Integer,minute,second).
timemeasure(60:Integer,minute,hour).
timemeasure(24:Integer,hour,day).
timemeasure(7:Integer,day,week).
timemeasure(3600:Integer,hour,second).
timemeasure(86400:Integer,day,second).
timemeasure(604800:Integer,week,second).
timemeasure(2592000:Integer,month,second).
timemeasure(31536000:Integer,year,second).

derivedunit(area, square_metre).
derivedunit(volume, cubic_metre).
derivedunit(density, kilograms_per_cub_metre).
derivedunit(speed, metre_per_second).
derivedunit(acceleration, metre_per_second_squared).
derivedunit(force,newton).
derivedunit(energy, joule).
derivedunit(molarity, mole_per_cubic_metre).
derivedunit(wavenumber, reciprocal_metre).
derivedunit(specificvolume, cubic_metre_per_kilogram).
derivedunit(currentdensity, ampere_per_square_metre).
derivedunit(magneticfield, tesla).
derivedunit(pressure, pascal).
derivedunit(power, watt).
derivedunit(electricpotential, volt).
derivedunit(electriccharge, coulomb).
derivedunit(capacitance, farad).
derivedunit(electricresistance, ohm).
derivedunit(electricconductance, siemens).
derivedunit(magneticflux, weber).
derivedunit(magneticfluxdensity, tesla).
derivedunit(inductance, henry).
derivedunit(radioactivity, becquerel).
derivedunit(specificenergy, joule_per_kilogram).
derivedunit(catalyticactivity, katal).
derivedunit(illuminance, lux).

lengthmeasure(1000:Real,kilometer,meter).
lengthmeasure(12:Real,foot,inch).
lengthmeasure(5280:Real,mile,foot).
lengthmeasure(1.09361:Real,yard,meter).
lengthmeasure(0.00497:Real,furlong,meter).
massmeasure(2.2046:Real,pound,kilogram).
massmeasure(0.001:Real,tonne,kilogram).
massmeasure(35.273:Real,ounce,kilogram).
massmeasure(0.06852:Real,slug,kilogram).

meter2kilometer(?Kilometer:Real,?Meter:Real) :-
                    lengthmeasure(?Value:Real,kilometer,?Unit),
                    basedimension(?Dimension,?Unit,m),
                    divide(?Kilometer:Real,?Meter:Real,?Value:Real).
```

```
meter2centimeter(?Centimeter:Real,?Meter:Real) :-
                    lengthmeasure(?Value:Real,centimeter,?Unit),
                    basedimension(?Dimension,?Unit,m),
                    multiply(?Centimeter:Real,?Meter:Real,?Value:Real).

meter2inch(?Inch:Real,?Meter:Real) :-
                    lengthmeasure(?Value:Real,inch,?Unit),
                    basedimension(?Dimension,?Unit,m),
                    multiply(?Inch:Real,?Meter:Real,?Value:Real).

meter2foot(?Foot:Real,?Meter:Real) :-
                    lengthmeasure(?Conversion:Real,inch,?Unit),
                    lengthmeasure(?Conversion1:Real,foot,inch),
                    basedimension(?Dimension,?Unit,m),
                    multiply(?Inch:Real,?Meter:Real,?Conversion:Real),
                    divide(?Foot:Real,?Inch:Real,?Conversion1:Real).

meter2mile(?Mile:Real,?Meter:Real) :-
                    lengthmeasure(?Conversion:Real,inch,?Unit),
                    lengthmeasure(?Conversion1:Real,foot,inch),
                    lengthmeasure(?Conversion2:Real,mile,foot),
                    basedimension(?Dimension,?Unit,m),
                    multiply(?Inch:Real,?Meter:Real,?Conversion:Real),
                    divide(?Foot:Real,?Inch:Real,?Conversion1:Real),
                    divide(?Mile:Real,?Foot:Real,?Conversion2:Real).

meter2angstrom(?Angstrom:Real,?Meter:Real) :-
                    lengthmeasure(?Conversion:Real,angstrom,?Unit),
                    basedimension(?Dimension,?Unit,m),
                    exponent(?Exponent:Real,angstrom),
                    pow(?Value,?Conversion:Real,? Exponent:Real),
                    multiply(?Angstrom:Real,?Meter:Real,?Value).

meter2yard(?Yard:Real,?Meter:Real) :-
                    lengthmeasure(?Conversion,yard,?Unit),
                    basedimension(?Dimension,?Unit,m),
                    multiply(?Yard:Real,?Meter:Real,?Conversion).

meter2furlong(?Furlong:Real,?Meter:Real) :-
                    lengthmeasure(?Conversion,furlong,?Unit),
                    basedimension(?Dimension,?Unit,m),
                    multiply(?Furlong:Real,?Meter:Real,?Conversion).

kilogram2gram(?gram:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,gram,?Unit),
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?gram:Real,?Kilogram:Real,?Value:Real).

kilogram2pound(?pound:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,pound,?Unit),
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?pound:Real,?Kilogram:Real,?Value:Real).

kilogram2tonne(?tonne:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,tonne,?Unit),
```

```
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?tonne:Real,?Kilogram:Real,?Value:Real).

kilogram2ounce(?ounce:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,ounce,?Unit),
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?ounce:Real,?Kilogram:Real,?Value:Real).

kilogram2milligram(?milligram:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,milligram,?Unit),
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?milligram:Real,?Kilogram:Real,?Value:Real).

kilogram2microgram(?microgram:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,microgram,?Unit),
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?microgram:Real,?Kilogram:Real,?Value:Real).

kilogram2slug(?slug:Real,?Kilogram:Real):-
                    massmeasure(?Value:Real,slug,?Unit),
                    basedimension(?Dimension,?Unit,kg),
                    multiply(?slug:Real,?Kilogram:Real,?Value:Real).

sec2min(?Min, ?Sec:Integer) :-
                    timemeasure(?Minute,minute,?Unit),
                    basedimension(?Dimension,?Unit,s),
                    integerDivide(?Mins, ?Sec:Integer, ?Minute),
                    mod(?Second, ?Sec:Integer, ?Minute),
                    inputToString(?Seconds,?Second),
                    inputToString(?Minutes,?Mins),
                    stringConcat(?Min, ?Minutes,  minute:String, ?Seconds,  second:String).

sec2hour(?Hrs, ?Sec:Integer) :-
                    timemeasure(?Hour,hour,?Unit),
                    basedimension(?Dimension,?Unit,s),
                    integerDivide(?Hr,?Sec:Integer,?Hour),
                    inputToString(?Hours,?Hr),
                    mod(?Rem,?Sec:Integer,?Hour),
                    sec2min(?Min, ?Rem),
                    stringConcat(?Hrs, ?Hours,  Hour:String, ?Min).

sec2day(?Dy, ?Sec:Integer) :-
                    timemeasure(?Day,day,?Unit),
                    basedimension(?Dimension,?Unit,s),
                    integerDivide(?D,?Sec:Integer,?Day),
                    inputToString(?Days,?D),
                    mod(?Remain,?Sec:Integer,?Day),
                    sec2hour(?Hour, ?Remain),
                    stringConcat(?Dy, ?Days,  Day:String, ?Hour).

sec2week(?Week, ?Sec:Integer) :-
                    timemeasure(?Wk,week,?Unit),
                    basedimension(?Dimension,?Unit,s),
                    integerDivide(?Wks,?Sec:Integer,?Wk),
                    inputToString(?Weeks,?Wks),
                    mod(?Remain,?Sec:Integer,?Wk),
```

```
                    sec2day(?Day, ?Remain),
                    stringConcat(?Week, ?Weeks,  Week:String, ?Day).

sec2month(?Mon, ?Sec:Integer) :-
                    timemeasure(?Month,month,?Unit),
                    basedimension(?Dimension,?Unit,s),
                    integerDivide(?Mont,?Sec:Integer,?Month),
                    inputToString(?Months,?Mont),
                    mod(?Remain,?Sec:Integer,?Month),
                    sec2day(?Day, ?Remain),
                    stringConcat(?Mon, ?Months,  Month:String, ?Day).

sec2year(?Year, ?Sec:Integer) :-
                    timemeasure(?Yr,year,?Unit),
                    basedimension(?Dimension,?Unit,s),
                    integerDivide(?Yrs,?Sec:Integer,?Yr),
                    inputToString(?Years,?Yrs),
                    mod(?Remain,?Sec:Integer,?Yr),
                    sec2month(?Mon, ?Remain),
                    stringConcat(?Year, ?Years,  Year:String, ?Mon).

min2hour(?Hrs, ?Min:Integer) :-
                    timemeasure(?Minute, ?Minu, hour),
                    basedimension(?Dimension, ?Unit, s),
                    divide(?Hrs, ?Min:Integer, ?Minute).

hour2day(?Dy, ?Hrs:Integer) :-
                    timemeasure(?Hour, ?Hr, day),
                    basedimension(?Dimension, ?Unit, s),
                    divide(?Dy, ?Hrs:Integer, ?Hour).

day2week(?Week, ?Days:Integer) :-
                    timemeasure(?Day, ?Dy,week),
                    basedimension(?Dimension, ?Unit, s),
                    divide(?Week, ?Days:Integer, ?Day).

Area(?Area:Real,?Length:Real, ?Unit) :-
                    derivedunit(area, ?Unit),
                    multiply(?Area:Real,?Length:Real,?Length:Real).

Volume(?Volume:Real,?Length:Real, ?Unit) :-
                    derivedunit(volume, ?Unit),
                    multiply(?Volume:Real,?Length:Real,?Length:Real,?Length:Real).

Density(?Density:Real,?Mass:Real,?Length:Real, ?Unit) :-
                    derivedunit(density, ?Unit),
                    Volume(?Volume:Real,?Length:Real,?UnitVol),
                    divide(?Density:Real,?Mass:Real,?Volume:Real).

Speed(?Speed:Real,?Length:Real,?Time:Real, ?Unit) :-
                    derivedunit(speed, ?Unit),
                    divide(?Speed:Real,?Length:Real,?Time:Real).

Acceleration(?Acceleration:Real,?Length:Real,?Time:Real, ?Unit) :-
                    derivedunit(acceleration, ?Unit),
                    multiply(?Timesq:Real,?Time:Real,?Time:Real),
```

```
        divide(?Acceleration:Real,?Length:Real,?Timesq:Real).

Force(?Force:Real,?Mass:Real,?Length:Real,?Time:Real, ?Unit) :-
        derivedunit(force, ?Unit),
        Acceleration(?Acceleration:Real,?Length:Real,?Time:Real,?UnitAcc),
        multiply(?Force:Real,?Mass:Real,?Acceleration:Real).

Energy(?Energy:Real,?Mass:Real,?Length:Real,?Time:Real, ?Unit) :-
        derivedunit(energy, ?Unit),
        Force(?Force:Real,?Mass:Real,?Length:Real,?Time:Real,?UnitForce),
        multiply(?Energy:Real,?Force:Real,?Length:Real).

Molarity(?Molarity:Real,?Mole:Real,?Length:Real, ?Unit) :-
        derivedunit(molarity, ?Unit),
        Volume(?Volume,?Length:Real,?UnitVol),
        divide(?Molarity:Real,?Mole:Real,?Volume).

WaveNumber(?WaveNumber:Real,?Length:Real, ?Unit) :-
        derivedunit(wavenumber, ?Unit),
        divide(?WaveNumber:Real, 1:Real,?Length:Real).

SpecificVolume(?SpecificVolume:Real,?Length:Real,?Mass:Real, ?Unit) :-
        derivedunit(specificvolume, ?Unit),
        Volume(?Volume,?Length:Real,?UnitVol),
        divide(?SpecificVolume:Real,?Volume,?Mass:Real).

CurrentDensity(?CurrentDensity:Real,?Ampere:Real,?Length:Real, ?Unit) :-
        derivedunit(currentdensity, ?Unit),
        Area(?Area,?Length:Real,?UnitArea),
        divide(?CurrentDensity:Real,?Ampere:Real,?Area).

MagneticField(?MagneticField:Real,?Ampere:Real,?Length:Real, ?Unit) :-
        derivedunit(magneticfield, ?Unit),
        divide(?MagneticField:Real,?Ampere:Real,?Length:Real).

Pressure(?Pressure:Real,?Mass:Real,?Length:Real,?Time:Real, ?Unit) :-
        derivedunit(pressure, ?Unit),
        Force(?Force:Real,?Mass:Real,?Length:Real,?Time:Real,?UnitForce) ,
        Area(?Area,?Length:Real,?UnitArea),
        divide(?Pressure:Real,?Force,?Area).

Power(?Power:Real,?Mass:Real,?Length:Real,?Time:Real, ?Unit)  :-
        derivedunit(power, ?Unit),
        Energy(?Energy:Real,?Mass:Real,?Length:Real,?Time:Real,?UnitEnergy),
        divide(?Power:Real,?Energy,?Time:Real).

ElectricPotential(?ElectricPotential:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real, ?Unit) :-
        derivedunit(electricpotential, ?Unit),
        Power(?Power:Real,?Mass:Real,?Length:Real,?Time:Real,?UnitPower),
        divide(?ElectricPotential:Real,?Power,?Ampere:Real).

ElectricCharge(?ElectricCharge:Real,?Time:Real,?Ampere:Real, ?Unit) :-
        derivedunit(electriccharge, ?Unit),
        multiply(?ElectricCharge:Real,?Time:Real,?Ampere:Real).

Capacitance(?Capacitance:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real, ?Unit) :-
```

```
                derivedunit(capacitance, ?Unit),
                ElectricCharge(?ElectricCharge:Real,?Time:Real,?Ampere:Real,?UnitCharge),
                ElectricPotential(?ElectricPotential:Real,?Mass:Real,?Length:Real,?Time:Real,
                ?Ampere:Real,?UnitPotential),
                divide(?Capacitance:Real,?ElectricCharge:Real,?ElectricPotential:Real).

ElectricResistance(?ElectricResistance:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real, ?Unit) :-
                derivedunit(electricresistance, ?Unit),
                ElectricPotential(?ElectricPotential:Real,?Mass:Real,?Length:Real,?Time:Real,
                ?Ampere:Real,?UnitPotential),
                divide(?ElectricResistance:Real,?ElectricPotential:Real,?Ampere:Real).

ElectricConductance(?ElectricConductance:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real,
?Unit) :-       derivedunit(electricconductance, ?Unit),
                ElectricPotential(?ElectricPotential:Real,?Mass:Real,?Length:Real,?Time:Real,
                ?Ampere:Real,?UnitPotential),
                divide(?ElectricConductance:Real,?Ampere:Real,?ElectricPotential:Real).

MagneticFlux(?MagneticFlux:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real, ?Unit) :-
                derivedunit(magneticflux, ?Unit),
                ElectricPotential(?ElectricPotential:Real,?Mass:Real,?Length:Real,?Time:Real,
                ?Ampere:Real,?UnitPotential),
                multiply(?MagneticFlux:Real,?ElectricPotential:Real,?Time:Real).

MagneticFluxDensity(?MagneticFluxDensity:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real,
?Unit) :-
                derivedunit(magneticfluxdensity, ?Unit),
                MagneticFlux(?MagneticFlux:Real,?Mass:Real,?Length:Real,?Time:Real,
                ?Ampere:Real,?UnitFlux),
                Area(?Area:Real,?Length:Real,?UnitArea),
                divide(?MagneticFluxDensity:Real,?MagneticFlux:Real,?Area:Real).



Inductance(?Inductance:Real,?Mass:Real,?Length:Real,?Time:Real,?Ampere:Real, ?Unit) :-
                derivedunit(inductance, ?Unit),
                MagneticFlux(?MagneticFlux:Real,?Mass:Real,?Length:Real,?Time:Real,
                ?Ampere:Real,?UnitFlux),
                divide(?Inductance:Real,?MagneticFlux:Real,?Ampere:Real).

RadioActivity(?RadioActivity:Real,?Time:Real, ?Unit) :-
                derivedunit(radioactivity, ?Unit),
                divide(?RadioActivity:Real,1:Real,?Time:Real).

SpecificEnergy(?SpecificEnergy:Real,?Mass:Real,?Length:Real,?Time:Real, ?Unit) :-
                derivedunit(specificenergy, ?Unit),
                Energy(?Energy:Real,?Mass:Real,?Length:Real,?Time:Real,?UnitEnergy),
                divide(?SpecificEnergy:Real,?Energy:Real,?Mass:Real).

CatalyticActivity(?CatalyticActivity:Real,?Time:Real,?Mole:Real, ?Unit) :-
                derivedunit(catalyticactivity, ?Unit),
                divide(?Activity,1:Real,?Time:Real),
                multiply(?CatalyticActivity:Real,?Activity,?Mole:Real).

ILLuminance(?ILLuminance:Real,?Length:Real,?Candela:Real, ?Unit) :-
                derivedunit(illuminance, ?Unit),
```

```
        Area(?Area,?Length:Real,?UnitArea),
        divide(?ILLuminance:Real,?Candela:Real,?Area).

fahrenheit2celsius(?Celsius,?Fahrenheit:Real) :-
        subtract(?Value:Real,?Fahrenheit:Real,32:Real),
        multiply(?Value1:Real,?Value:Real,5:Real),
        divide(?Celsius:Real,?Value1,9:Real).

celsius2fahrenheit(?Fahrenheit,?Celsius:Real) :-
        multiply(?Value:Real,?Celsius:Real,9:Real),
       divide(?Value1:Real,?Value:Real,5:Real),
       add(?Fahrenheit:Real,?Value1,32:Real).
```

9.2. Built-in to convert Input to String

Only string literals can be concatenated in the current version of OOjDREW, we should typecast integer or any other data type into string in order to concatenate; to overcome this issue we have developed "InputToString" built-in e.g. concatenating integer 1 and string hour , integer 1 is type casted to string before concatenation.

```
package jdrew.oo.builtins;
import java.util.*;
import java.io.*;

import jdrew.oo.util.DefiniteClause;
import jdrew.oo.util.SymbolTable;
import jdrew.oo.util.Term;
import jdrew.oo.util.Types;

public class InputToString implements Builtin {

 public String c;
 public Integer d;
 public String p2d;
 public String p2s;

private int symbol = SymbolTable.internSymbol("inputToString");

 public DefiniteClause buildResult(Term t) {
 if (t.getSymbol() != symbol) {
 return null;
 }
 Term p2 = t.subTerms[2].deepCopy();
 if (p2.getSymbol() < 0) {
 return null;
 }

 if (p2.getType() == Types.IFLOAT || p2.getType() == Types.IINTEGER ||
p2.getType()==Types.ISTRING ||p2.getType()==Types.INUMERIC
||p2.getType()==Types.IOBJECT || p2.getType()==Types.INOTHING
||p2.getType()==Types.ITHING) {

 p2s = p2.getSymbolString();
```

```
 try {
 p2d = p2s.toString();
 } catch (Exception e) {
 return null;
 }
 }
 String result = p2d;
 String results = result.toString();
 Term r1 = new Term(SymbolTable.internSymbol(results),
 SymbolTable.INOROLE, Types.ISTRING);
 Term roid = new Term(SymbolTable.internSymbol("$jdrew-inputToString-" + p2s),
 SymbolTable.IOID, Types.ITHING);

 Vector v = new Vector();
 v.add(roid);
 v.add(r1);
 v.add(p2);
 Term atm = new Term(symbol, SymbolTable.INOROLE, Types.IOBJECT, v);
 atm.setAtom(true);
 Vector v2 = new Vector();
 v2.add(atm);
 return new DefiniteClause(v2, new Vector());
 }

 public int getSymbol() {
 return symbol;
 }
}
```

9.3. Website URL

http://specifying-units-of-measure.yolasite.com/