STANDARDIZATION OF PHYSICS EXERCISES: DYNAMICAL GENERATION OF DATA

Joan Parellada

Standardization of Physics Exercises. Dynamical Generation of Data.

J. Parellada
Facultat de Física
Universitat de Barcelona
Diagonal 645
08028 BARCELONA SPAIN

ABSTRACT

Problems are one the most commonly used methods in the study of physics; thus computer-assisted tools to promote their use should be welcomed. To make a problem reusable under different computer environments, its contents must be structured in a standard way so that any code can understand it. The structure and tags defined in IMS QTI specifications are used and new tags are introduced to make the best of the tutoring capabilities of a problem. Furthermore, in order to have better control of students' activities, the problem data should be made variable. The idea of a problem template is introduced and a set of tags to dynamically generate the data is presented.

INTRODUCTION

Problems and exercises are normally presented to students in physics and other scientific areas as a self-learning tool. It is accepted that just trying to solve them is a good method to master the concepts involved in science, concepts that have been normally explained previously in classrooms or textbooks. Teachers and lecturers believe that the ability to solve an exercise is strongly related to student knowledge, so exercises are often presented in exams to grade the students.

The problems are usually corrected and graded by lecturers or tutors following a lengthy process. All the assumptions, steps and calculations done by the students have to be checked and understood by the reviewer, looking for misunderstandings and/or mistakes. Although quite often the whole development leads to a final result, sometimes a number, it is not good teaching practice just to check this number to assess all of the work done by the student; a small error in a calculation is not the same as a critical misunderstanding of a concept. On the other hand, if the final result is correct, one can assume development to be right; normally a series of mistakes never compensates errors to give the correct value.

For all these reasons, except in non-trivial exercises, it is difficult to automate the correction process. It could be done if the whole exercise is split into small steps with some sequencing. Unfortunately if fixed sequencing is used, the pathway can also be seen as a hint on how to proceed. Adaptive sequencing can solve this shortcoming, but it is much more difficult to implement.

There is an important point to be raised in Latin cultures, which may not be strictly true in more Anglo-Saxon environments. It has to do with the need to strictly follow the activities of the students and a false concept of solidarity among them. One can imagine making it mandatory to try to solve a problem and to record student's activities, but in distance learning it is too easy for students to pass information to each other and to cheat that student overall activity can not be trusted. If each one of the students were presented with their own exercise, or at least with different data, it is expected that personal work to get the numerical answer can be ensured, because no student will want to repeat calculations just so they can be passed to his or her fellow students.

More specific details have also to be considered when dealing with problems or exercises in physics. Numerical results should take into account accuracy and precision issues. Although Merriam-Webster (2003) defines them in a very similar way, actually to be synonymous, they are different concepts which lead to considerable confusion among students. Accuracy is related to the correctness of the result; normally different numerical calculations lead to different results because of rounding errors, and should be taken into account given an upper and a lower limit for a correct value. Precision is related to how exactly the value is computed, the number of significant digits given in the result is an indication of the precision.

Another issue has to do with the fact than normally a result is not just a number, but a physical magnitude, which means a number plus the unit qualifying it. Thus there are different ways to write the same result; a different figure corresponding to a different unit. Once more, it does not seem to be good teaching practice to ask the student to use a particular unit; normally there is a more adequate one, and being able to find it is also part of the problem, and an intrinsic process in learning physics.

```
<html>
<head>
<title>Problema Mecanica 01_01F_01_EN</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<?php
  include ("c:web_exemples\problemes\php\comu1.php");
  $units = array("hours", "km", "m", "miles", "yards", "lux");
  $conversion = array(array("na",0), array(1,0), array(1000,0),
       array(0.6213881812968,0) ,array(1093.61329833771,0),
       array("na",0));
  $speed = number_format(doubleval(rand(20,100)),0);
  $time  = number_format(doubleval(rand(10,120)),0);
  $precision = min(precision($speed), precision($time))+1;
  $answer = $speed*$time/60;
  $enunciat = "A car runs at $speed km/h. What distance will it
     travel in $time minutes?";
  $pregunta = "The distance is  ";
   hhint = array (array("text","Recall that distance equals speed
     times time",0), array("text","Have you changed the minutes
     to hours, dividing by 60?",0));
   include ("c:web_exemples\problemes\php\comu2.php");
?>
</body>
</html>
```
Fig 1.- PHP computer code

A PHP and MySql computer code has been developed to present physics exercises following this approach. Dynamical generation of numerical application and mathematical formulation of the solution is programmed. Units and precision issues are taken into account. Hints can be supplied, and a particular sequencing and pace for the problem set can be dynamically generated for each individual

To write down a full exercise this way is quite a complex procedure. First, text has to written; then range and/or values of data and their presentation have to be analyzed to avoid mathematical singularity computations or physically nonsensical results; later on, the mathematical expressions leading to the final results have to be correctly coded; furthermore, hints have to be prepared in relation to foreseen wrong answers.

As is seen, each problem is a PHP file, with all the exercise characteristics defined as computer variables placed between two include sentences, they call other files containing all the logic. In any case, there is a very close binding between the code and the problems. To be efficient, exercises can not just be "variables" closely related to a computer code, they would have to be rewritten every time changes in the code are made or could not be used at all in any other program. Our whole goal is to consider a problem as a *"learning object"* (**LO**) as defined by D.A.Wiley (Wiley 2002) *"any digital resource that can be reused to support learning"*.

META-DATA

In this paper we propose a meta-information dataset related to problems in physics in general. This information deals with classification, authoring and versioning, common to any **LO,** to a more specific approach for problems or exercises.

The first set is clear; there has been substantial work already done on digital content, in general, and learning material, in particular. Dublin Core (DC 2003), IEEE/LTSC (IEEE 2002) and IMS LOM (IMS 2001), but there is still much to do: the tag set has to be increased, for example, to include some information on translations (CEN CWA 14645 2003), and controlled vocabularies should be established and maintained for many subjects, education and physics in particular (CEN/ISSS Draft CWA 2003).

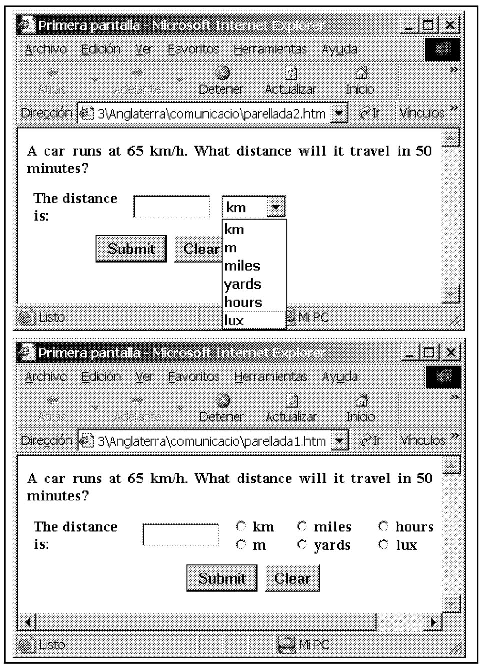
The specific tag set for problems has already been worked by IMS in Question & Test Interoperability **(QTI)** specifications (IMS/QTI 2002) although it is not well adapted to our requirements. The main objectives required by **QTI** are already explained in its vocabulary. The specifications define Assessment, Section and Item. It should be clear then that the main objective of **QTI** specifications is interoperability among tests (equivalent to assessments) made up of questions (equivalent to items) that can be grouped or not in sections. In our vocabulary, we have a problem or exercise which normally has a general statement followed by one or more ordered questions; each one could have just one clause or more (basic or composite response type in **QTI** notation). The student should supply the answer(s) to the question, normally via the submit/enter function, and the given values are checked against known ones. The sequencing rules will determine the next question (item in **QTI** nomenclature) to be presented. For coherence with IMS **QTI** specifications, our problem will be a section containing a set of items. Normally the questions will be closely related to the main statement of the problem and that will set down the granularity at a section level.

PROPOSED NEW RESPONSE TYPES

The main presentation elements defined in **QTI** are related to defining the question type, how it is rendered and how the answer is to be given. The basic answers defined are: Logical Identifier, XY co-ordinate, string, numerical and logical group, and each one of them can have different ways of being presented. String and numerical groups are what they mean; a logical identifier is related to choices from a set of values and a logical group is used to provide relations between different sets of values, for example using drag and drop. With XY co-ordinates, HTML image maps can be used as presentation material and an area or point is chosen in the student's answer.

In physics we can use all these different ways to present questions; actually a quite simple astronomy problem is the example that **QTI** presents for the logical group; to name and order all the planets of our solar system. This means that rendering a specific **QTI**-defined element will always be the same and coherent with the specifications. Rendering will be mainly determined by the processing code using the XML file as data. As an example, if choices are to be presented as radio button or a checkbox, depending on whether only one or more than one answer is expected, then the select element allowing the student to choose from a scrolled list will never appear.

Probably the most typical answer in physics problems would be a number with physical units. Although this response could be obtained using a composite item with a numerical field plus a single choice, for different reasons it seems appropriate to introduce a new response type. The answer should give a numerical quantity, the unit and the precision, if it has to be considered, thus leading to three response labels. Furthermore, there is a presentation problem because while the numerical response will normally be rendered by one input box, fill in a blank **(FIB)** in **QTI** nomenclature, the unit is a choice normally presented as a radio button field. Although pedagogically correct, it is an ugly way to write the result, essentially because if it were presented as a box followed by a select field with size one, it would look much more normal.


Primera pantalla - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Adelante Detener Actualizar Inicio
Dirección 3\Anglaterra\comunicacio\parellada2.htm Ir Vínculos
Submit
Clear
Listo
Mi PC
Primera pantalla - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Adelante Detener Actualizar Inicio
Dirección 3\Anglaterra\comunicacio\parellada1.htm Ir Vínculos
Submit
Clear
Listo
Mi PC

A car runs at 65 km/h. What distance will it travel in 50 minutes?

The distance is: km — km, m, miles, yards, hours, lux

A car runs at 65 km/h. What distance will it travel in 50 minutes?

The distance is: ○ km ○ miles ○ hours ○ m ○ yards ○ lux

Fig 2.- Rendering

If the author does not wish to give any hint on the units to be used, one or two **FIB** fields can also be utilized. The processing code will then have to analyze the input to find the corresponding unit, which is more prone to errors.

In composite items it is expected that each one of the entered values should be checked against those given. In our case, there is a unit transformation factor that relates the figure with the unit, the same result could be written in different ways; for example 1000mm = 100cm = 1m = 0.001km. Then, either equivalent values are given in the response processing section, or the student response is transformed by the code. Factors should be given as data to change the answer to the same unit used by the author, and later both numerical values are directly compared. It could be worthwhile to offer a unit name that is not applicable for the desired solution. it should be decided how to mark these units, a factor “0” or a “NA” attribute, setting a pass/fail flag in processing the answer, or just passing all the possibilities to the response analysis. We shall take the last case as an example.

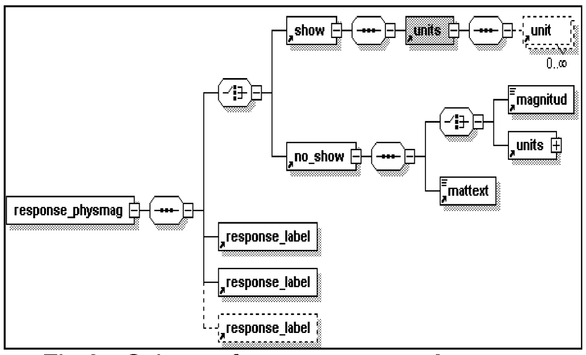


Fig 3.- Schema for **<response_physmag>**

A new response type to take into account all these details is presented, its XML representation is shown and an example offered. The three simultaneous answers expected as input from the student are differentiated by the attribute *ident* of the **<response_label>** tag. Its value can reasonably be set to *value*, *unit*, and *precision*. If precision is not to be analyzed, then just do not write the corresponding **<response_label>** tag.

THE RESPONSE ANALYSIS

The **<resprocessing>** tag has to be used to analyze the answer given by the student, essentially the **<respcondition>** sub-element that should appear one or more times inside it. To analyze an answer given by a number and a unit, both will have to be checked, and it is up to the author to make one of them exclusive using the *continue* attribute in **<respcondition>**. It will also be the time to check for precision, either by giving a positive integer, meaning the maximum number of digits the answer can take, or using a default value obtained from the data used in the calculation.

```
<item ident="s=vt">
  <presentation>
    <material> A car runs at 65 km/h. What distance will it
          travel in 50 minutes </material>
    <response_physmag ident="distance">
      <show>
        <units shuffle="yes">
          <unit name="km" factor="1.0" shift=0.0"/>
          <unit name="m" factor="1000." shift=0.0"/>
          <unit name="miles" factor="0.6213881812968"/>
          <unit name="yards" factor="1093.61329833771"/>
          <unit name="hours" factor="0.0" shift=0.0"/>
          <unit name="lux" factor="0.0" shift=0.0"/>
        <units/>
      <show/>
    <response/>
    <response_label ident="unit"/>
    <response_label ident="value"/>
    <response_label ident="precision"/>
  <presentation/>
<item/>
```

Fig 4.- XML Instance exercise

As an example, the instance to analyze the example above is presented.

```
<resprocessing>
<respcondition  title="Correct" >
  <conditionvar>
   <and>
    <or>
     <varequal respident="unit">km</varequal>
     <varequal respident="unit">m</varequal>
     <varequal respident="unit">miles</varequal>
     <varequal respident="unit">yards</varequal>
    <or>
    <varequal respident="precision">4</varequal>
    <varlte respident="value">54.2</varequal>
    <vargte respident="value">54.1</varequal>
   </and>
  </conditionvar>
  <displayfeedback feedbacktype="Response" linkrefid=
        "correct"/>
 </respcondition>
 <respcondition continue="No" title="units_error" >
  <conditionvar>
   <or>
    <varequal respident="unit">hours</varequal>
    <varequal respident="unit">lux</varequal>
   <or>
  </conditionvar>
  <displayfeedback                    feedbacktype="Hint"
linkrefid="u_err"/>
 </respcondition>
 <respcondition continue="No" title=" low_precision">
  <conditionvar>
   <varlt respident="precision">4</varequal>
  </conditionvar>
  <displayfeedback feedbacktype="Hint" linkrefid="l_p"/>
</respcondition>
 <respcondition continue="No" title=" high_precision" >
  <conditionvar>
   <vargt respident="precision">4</varequal>
  </conditionvar>
  <displayfeedback feedbacktype="Hint" linkrefid="h_p"/>
 </respcondition>
 <respcondition continue="No" title="value_error" >
  <conditionvar>
   <or>
    <varlt respident="value"> 54.1 </varequal>
    <vargt respident="value">54.2 </varequal>
   <or>
  </conditionvar>
  <displayfeedback feedbacktype="Hint" linkrefid="v_err"/>
 </respcondition>
</resprocessing>
<itemfeedback ident="u_err">
 <material> Wrong units </material>
</itemfeedback>
<itemfeedback ident="l_p">
 <material> Please, write more digits </material>
</itemfeedback>
<itemfeedback ident="h_p">
 <material> Too many digits!!! </material>
</itemfeedback>
<itemfeedback ident="v_err">
 <material> Wrong numerical answer!. </material>
</itemfeedback>
<itemfeedback ident="correct">
 <material> Congratulations, right answer!!! </material>
</itemfeedback>
```

Fig 5.- XML instance **<respcondition>**

QTI specifications allow feedback information to be assigned to each **<responcondition>** tag. This information is presented to the student if the condition result is true. A message congratulating the student can be sent if the correct value is introduced or, on the other hand, hints can be presented to help him or her to keep trying to solve the question. Typical errors can also be checked and more specific hints can be given for each one.

It is sometimes reasonable to present the student who has not answered a question correctly with another one, simpler and easier. In our example, a typical mistake could be just to forget to express the time in hours before it is multiplied by velocity. In this case presenting an auxiliary question, "*how many hours correspond to 50 minutes?",* could help the student to understand and solve the problem. IMS **QTI** has recently issued specifications for Selection and Ordering (IMS/QTI-SO 2002), but the user case we wish to include is still being studied. Probably in the future, complete, deep relations could be established between questions and answers; in the meantime our proposal will be to add a new sub-element into **<responcondition>** to jump to a specified item. This can be done in a similar way to how the feedback is presented, it is mandatory that each of the items has a unique *ident* attribute, and to make it simple the proposed **<jump>** element should only have a *linkrefid* whose value corresponds to the next item to be presented.

THE META-PROBLEM

In **QTI** specifications there is no feature for having variable data and/or text in the material element of assessment, sections or items. This means that the XML instance fully determines the rendering of the material and the evaluation of answers without any other process. As said in the introduction, one of our goals was to be able to generate different numerical applications, which lead to different results, in the same problem; this is the concept of a *problem template*.

Two different approaches can be used to work out development. In one, all the information is placed in the same **XML** instance and processed by the same code. First, it should hold all the data on how to define and assign values to the variables, later followed by **QTI**-style structure. In the second part, the variable elements should be defined inside a **XML** *Processing Instruction* **PI,** which we can call *replace*, to let the code know that its *targetname* is just the name of a previously defined and assigned variable. The variable value has to be placed in the precise position where the **PI** is located. How the code does this replacement is not defined here.

```
<mattext>
    A car runs at <?replace  v ?> km/h. What distance
     will it travel in <?replace t ?> minutes?
</mattext>
```

FIG 6.-**PI** instruction use

This approach is not the best practice in XML, where **PI** is not in the spirit of generalized markup, and is not recommended in IMS specifications, where mixed content elements are not allowed.

The second approach is modular. Normally, in a **QTI** instance, all problem statements and results are just **PCDATA** within some elements. Our proposal for a **QTI**-style problem template just adds two new elements to split this data into a fixed part and a variable part. The fixed element **<fix>…</fix>** will again hold data and the variable element **<var>…</var>** will just name the variable defined elsewhere.

To proceed, first all information related to names and values of variables has to be given. We can use an **XML** instance, but one that is not related to **QTI,** or any other data file as input to a computer code. The output is a listing with the variable names and its associated value. Next, the **XML** file defining the problem template is parsed and transformed, the use of **XSLT** will probably represent an overkill, tags **<fix>** and **</fix>** are removed and the character set **<var>***var_name***</var>** is replaced by the *var_name* value from the listing defined previously. The file obtained from the transformation will be a regular **QTI** file that could treated by any code following **QTI** specifications.

```
<mattext>
   <fix>A car runs at </fix><var> v</var><fix> km/h. What distance
    will it travel in </fix><var> t</var><fix>minutes?</fix>
</mattext>
--------------------------------------------------------------------
                              v = 65
                              t = 50
--------------------------------------------------------------------
<mattext>
   A car runs at 65 km/h. What distance will it travel in 50 minutes?
</mattext>
```

Fig 7.- Substitution procedure

The problem template should be cataloged as a regular **LO**, within a package in **IMS** nomenclature, which needs an auxiliary file containing the information related to variable generation. It should be up to the Learning Managing System code to generate the **QTI** instance on the go, for each request, or to build and store a set of XML files to be presented to different students.

A XML BINDING FOR VARIABLE DEFINITION

A distinction has to be made between the generation of data and the computation of results, both are variables that have to be defined; the first could have a random process, while the second is fully deterministic. There will be then two main elements, **<var_gen>** to generate data and **<var_com>** to compute results.

There could sometimes be a need to randomize no numerical data, pure text or units, associated with numerical data. The **<var_gen>** element should have sub-elements named **<option>**, with an attribute called *weight*. Inside **<option>** all the variables should be defined, but only set will be chosen, using the attribute as a weight.

Within **<option>,** we can have **<var_text>** elements defining the name of the variable and its value. We shall consider three different kinds of numerical variables: *Single*, which is just a single number, defined from a given value, or sampled from a list, or randomly generated from a set of intervals, or obtained from a mathematical computation of previously defined variables. We can also define *Related* variables, a set of variables which are sampled from a list in a correlated way, the i-th element of each list is assigned to its variable name. Finally, we can have just one *Multiple* variable, a variable name assigned to a set of values, an array in computer language, obtained either from a matrix or from a computation.

A **<var_cal>** sub-element, containing only a mathematical computation, must also be included in **<option>** to make data uniform among options.

Each of these elements must have an attribute defining how the number has to be stored in memory, as an integer or float or double precision, and a sub-element defining how to write it, as an integer, decimal or in scientific notation, giving information on the number of total digits and the number of decimals. A prevision to download a file should be included in a multiple variable.

In the **<var_com>** element we can only have **<var_cal>** sub-elements containing the variable name, the mathematical expression defining them and information on their representation. Different approaches are being considered for mathematical computations. Fully reusable MathML sentences can be used, but the code using the instance has to be able to understand the language. The easiest approach, at least for simple calculations, can be implemented using a **PI** instruction **<? eval** *expression***?>**; almost all languages have an *eval* sentence and an easy way to write simple mathematical expressions. For more complex calculations, **CDATA** can be used to pass the code to the program; to make this approach reusable, the **CDATA** should be placed within a container with the name of the language used and various containers can be placed in the instance.

The XML instance to randomize our example is listed. The XML schema definition can be requested from the author.

```xml
<var_gen>
  <option name="car" weight="5">
    <var_text name="text"> A car runs at </var_text>
    <var_text name="speed_unit"> km/h </var_text>
    <var_text name="time"> minutes </var_text>
    <var_num_single name = "v" type = "int">
      <random>
        <interval>
          <value> <cte>70</cte> </value>
          <value> <cte>95</cte> </value>
        </interval>
      </random>
      <presentation type="int">
        <digits>2</digits>
       </presentation>
    </var_num_single>
    <var_num_single name = "t" type = "int">
      <list> 40,45,50,55,65 </list>
    </var_num_single>
    <presentation type="int">
      <digits>2</digits>
    </presentation>
    <var_cal name="v_ana" type = "float">
      <?eval $v_ana=$v?>
    </var_cal>
    <var_cal name="t_ana" type = "float">
      <?eval $t_ana=$t/60?>
    </var_cal>
  </option>
  <option name="plane" weight="3">
    <var_text name="text"> A plane flights at </var_text>
    <var_text name="speed_unit"> miles/h </var_text>
    <var_text name="time"> minutes </var_text>
    <var_num_single name = "v" type = "int">
      <random>
        <interval>
          <value> <cte>500</cte> </value>
          <value> <cte>600</cte> </value>
        </interval>
      </random>
      <presentation type="int">
        <digits>3</digits>
      </presentation>
   </var_num_single>
   <var_num_single name = "t" type = "int">
      <list> 55,65,85</list>
    </var_num_single>
    <presentation type="int">
      <digits>2</digits>
    </presentation>
    <var_cal name="v_ana" type = "float">
      <?eval $v_ana=1.6093*$v?>
    </var_cal>
    <var_cal name="t_ana" type = "float">
      <?eval $t_ana=$t/60?>
    </var_cal>
  </option>
  <option name="snail" weight="2">
    <var_text name="text"> A snail creeps at </var_text>
    <var_text name="speed_unit"> cm/minute
</var_text>
    <var_text name="time"> hours </var_text>
    <var_num_single name = "v" type = "int" >
      <random>
        <interval>
          <value> <cte>9</cte> </value>
          <value> <cte>18</cte> </value>
        </interval>
      </random>
      <presentation type="int">
        <digits>2</digits>
      </presentation>
    </var_num_single>
    <var_num_single name = "t" type = "float">
      <list> 0.6, 0.8, 1.2 </list>
    </var_num_single>
    <presentation type="dec">
      <digits>2</digits><dec>1</dec>
    </presentation>
    <var_cal name="v_ana" type = "float">
      <?eval $v_ana=6*$v/10000?>
    </var_cal>
    <var_cal name="t_ana" type = "float">
      <?eval $t_ana=$t?>
    </var_cal>
  </option>
</var_gen>
<var_com>
  <var_cal name="dis_correct" type = "float">
    <?eval $dis_correct=$v_ana*$t_ana?>
  </var_cal>
  <var_cal name="dis_wrong1" type = "float">
    <?eval $dis_wrong1=$v_ana*$t?>
  </var_cal>
  <var_cal name="dis_wrong2" type = "float">
    <?eval $dis_wrong2=$v*$t_ana?>
  </var_cal>
  <var_cal name="dis_wrong3" type = "float">
   <?eval $dis_wrong3=$v*$t?>
  </var_cal>
</var_com>
```

Fig 8.- Variable data definition

FUTURE WORK

To check the suitability of the proposed structure, a small set of problems from different subjects in physics is being built by specialists. Probably new response types will be needed, a few of them have already been suggested: symbolic mathematical expression, vector analysis, universal unit definition and transformation files,…

The PHP code is already being adapted to use XML instances using SAX libraries, although DOM could be more suitable.

An authoring program to lead and help authors in writing the XML file is also planned.

REFERENCES

CEN/ISSS CWA 14645 (2003) *Availability of alternative language versions of a learning resource in IEEE LOM* <http://www.cenorm.be/isss/cwa_download_area/cwa14645.pdf>

CEN/ISSS (2003) *Draft CWA Controlled Vocabularies for Learning Object Metadata* <http://office.eun.org/kms/sites/cenisss/TaxVocReport_1_3.zip>

Dublin Core *Metadata Element Set, Version 1.1: Reference Description* (2003) <http://dublincore.org/documents/dces>

IEEE 1484.12.1-2002 *Final Draft Standard for Learning Object Metadata (2002)* <http://ltsc.ieee.org/doc/wg12/LOM_1484_12_1_v1_Final_Darft.pdf>

IMS Learning Resource Meta-data XML Binding, Ver.1.2.1 (2001) <http://www.imsglobal.org/metadata/imsmdv1p2p1/imsmd_bindv1p2p1.html>

IMS Question & Test Interoperability: ASI XML Binding, Ver.1.2 (2002) <http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_bindv1p2.html>

IMS Question & Test Interoperability: ASI Selection & Ordering, Ver.1.2 (2002) <http:// www.imsglobal.org/question/qtiv1p2/imsqti_asi_saov1p2.html>

Merriam-Webster On Line <http://www.m-w.com/home.htm>

Wiley D.A. (2002) *Connecting learning Objects to instructional design theory: A definition, a metaphor, and a taxonomy in The Instructional Use of Learning Objects* <http://reusability.org/read/>