Project no. 018340

Project acronym: EDIT

Project title: Toward the European Distributed Institute of Taxonomy

Instrument: Network of Excellence

Thematic Priority: Sub-Priority 1.1.6.3: "Global Change and Ecosystems"

C5.132 Full implementation of CDM Library functionality for the output of structured descriptive content in structured and textual form

Due date of component: Month 52
Actual submission date: Month 52

Start date of project: 01/03/2006 Duration: 5 years

Organisation name of lead contractor for this component: 2 MNHN

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

INTRODUCTION

The CDM library offers a lot of elements and functionalities in order to help taxonomists store their data.

However, at least two elements were missing: the possibility to generate a clear and easy to read output of these data, either for publication or simply to have a readable preview of a part of or even an entire database; the possibility to automatically generate identification keys from these data, which is an essential goal of taxonomists, since taxonomy is the science of classification.

The first functionality is now possible thanks to the "natural language" objects and functions. Moreover, the possibility to predefine the structure of the output allows the scientists to keep a constant scheme which is necessary when writing monographs which are standardized and thus save a quite important amount of time.

Key generation functions are available through the IdentificationKeyGeneration class. As for natural language descriptions, these keys can help experts as much as they can be used by unskilled people, depending of course on the complexity of the key and the module developed is thus a core back end of the CDM library.

TECHNICAL ACHIEVEMENT

A) Natural language

1. Generalities

The minimum arguments to provide to generateNaturalLanguageDescription(), the basic function of the class NaturalLanguageGenerator, are a FeatureTree and a TaxonDescription: the FeatureTree containing the order in which the different elements (Set<DescriptionElementBase>) of the TaxonDescription will be read. The principle is the following:

- the function walks the tree and checks if the feature associated to the current FeatureNode matches the one of a DescriptionElementBase contained in the TaxonDescription. If such an element is found a DescriptionBuilder is called. Two possibilities: either it is a QuantitativeData which calls a QuantitativeDescriptionBuilder or it is a CategoricalData which calls a CategoricalDescriptionBuilder. NaturalLanguageGenerator complete at each step a List<TextData>, which in the end is returned, corresponding to the full description of the given taxon.

The possibilities of this system are numerous, one can for example generate as many descriptions as TaxonDescription in the database once the tree has been created (which is quite important because monographs are standardized for publications) and several trees can be used to produce different descriptions from the same TaxonDescription, depending on their purpose.

2. Language

Two other versions of the function generateNaturalLanguageDescription are also available, with an extra argument, either a Language or a List<Language>. In the first case, the description will be generated in the specified language, and if, for an element, no description can be found in this language, the method falls back to return the default Representation (getPreferredRepresentation() method of the TermBase class). In the second case, the way to operate is the same except the function will try to find a representation in all the different languages of the List, using the order in which they are specified.

3. Output format

The DefaultDescriptionBuilders returns the description in simple text format whereas, for example, the MicroFormatDescriptionBuilders returns a description in XHTML custom MicroFormat (http://en.wikipedia.org/wiki/Microformat), for instance:

```
<span class=”state”>from <span class=”measurement”>3.0</span> to <span class=”measurement”> 6.0</span>
<span class=”unit”> mm</span></span>
```

Moreover, the path of the feature in the tree is also printed:

<span class=”feature cauline_leaves-lamina-legth”>….

All these specificities make the resulting description highly customizable, thanks for example to a CSS style sheet, one can for example, chose to define different kinds of paragraphs depending on the depth of the feature in the tree, or use different colors or styles for measures, modifiers, and so on.
Finally, it is also possible to create other specific DescriptionBuilders if needed, allowing to produce descriptions in all kind of formats.

4. Example

Here is a key created from the file "cichorieae.sdd" in copy with this component.

lower cauline leaves in outline obovate. lower cauline leaves lamina length from 5.0 cm. lower cauline leaves lamina width from 1.0 cm. lower cauline leaves incision pinnatifid irregularly runcinate lyrate. lower cauline leaves margin entire denticulate lacerate. lower cauline leaves apex acute obtuse. lower cauline leaves base petiole-like attenuate. lower cauline leaves indumentum subglabrous hispidulous. peduncle length (exact) up to 20.0 cm. peduncle shape slightly swollen. peduncle indumentum glabrous hispidulous. involucral bracts colour (if concolourous) green purplish. involucral bracts indumentum shortly tomentose glandular. outer involucral bracts shape to ovate linear. inner involucral bracts shape ovate linear. inner involucral bracts length from 10.0 mm. inner involucral bracts width from 1.0 mm. inner involucral bracts apex subacute. inner involucral bracts indumentum pubescent. inner involucral bracts midrib strong, prominent. corolla shape ligulate. corolla colour yellow.

B) Identification keys

1. Structure

Identification keys are represented as the CDM object PolytomousKey which extends FeatureTree: these keys have indeed a tree structure each node representing a question about a feature and each link an answer to this question, being the corresponding state. For example, a node will hold the question "What is the color of the wings ?" and the links will correspond to "white", "red", etc. To each node, a set of taxa can be associated, representing all the taxa for which the states match the path leading to this node.

2. Generalities

The algorithm used in IdentificationKeyGenerator is the base to create identification keys, there are many ways to complicate the process, depending on the users need. Here is the architecture of the recursive function, buildBranches(), used to build the keys:

- its arguments are the FeatureNode corresponding to the previous "best" Feature, the list of features left i.e. that have not won previously, the taxa concerned by the feature considered and finally the taxon/feature matrix which holds all the data for the computation.
- the first operation is the calculus of the scores of each remaining features, with the function FeatureScores() which calls FeatureScore(). This one gives the score of a single feature with the taxa concerned.
- This score, by default (method DefaultPower()), corresponds to the discriminating power of a feature which is, in this case, calculated the following way: if two taxa have no state in common (trivial for qualitative data, variable for quantitative data) the score is incremented of 1 else nothing is done. The resulting score is thus the sum for all the possible pairs of taxa. Of course, other functions can be easily added to replace this score.
- the second operation is then to determine which is the "winning" feature, i.e. the feature which we consider has the best score. By default, the function DefaultWinner() returns this feature, by giving the first feature to have the closest score to the "best possible score"*.
- in the third place, for each possible state for the winning feature, a node is created and attached to the previous best feature passed in argument of the function

- then, for each of these nodes, specific functions determine the new remaining taxa, and features and the loop continues until the taxa are no more discriminated (i.e. there is only one left or several taxa can not be discriminated any more, the node becomes a leaf)

buildBranches(), is initialized with the root of the PolytomousKey (automatically instantiated when creating a FeatureTree) and the whole sets of TaxonDescriptions and Features.

*this algorithm tends to minimize the mean number of nodes in the key.

3. Example

Here is a key created from the file "pinus.sdd" in copy with this component.

1 : 1 Nombre d'aiguilles par faisceau 2
- 2 : 1 Forme de l'extrémité des bourgeons obtuse
  - 3 : 1 couleur orange
  - 3 : 2 couleur brun
  - 3 : 3 couleur gris
  - 3 : 4 couleur vert
- 2 : 2 Forme de l'extrémité des bourgeons pointue
  - 4 : 1 Longueur des graines <5 cm
    - 5 : 1 Aspect de la surface des rameaux jeunes luisant
      - 6 : 1 Symétrie du cône symétrique
      - 6 : 2 Symétrie du cône dissymétrique
    - 5 : 2 Aspect de la surface des rameaux jeunes mat
      - 7 : 1 Symétrie du cône dissymétrique
      - 7 : 2 Symétrie du cône symétrique
  - 4 : 2 Longueur des graines de 5 à 10 cm
    - 8 : 1 Aspect de la surface des rameaux jeunes luisant
      - 9 : 1 Symétrie du cône symétrique
      - 9 : 2 Symétrie du cône dissymétrique
    - 8 : 2 Aspect de la surface des rameaux jeunes mat
      - 10 : 1 Symétrie du cône dissymétrique
      - 10 : 2 Symétrie du cône symétrique

- ....

CONCLUSION

The natural language module is now fully integrated to the CDM library and allows users to easily and rapidly generate a monograph describing the specimens, taxa (species, etc) of their choice. Using FeatureTrees allows this operation to be reiterated even faster once at least one tree has been created: the descriptions are based on the same model, which in the end saves a lot of time of the taxonomist. Moreover, the possibility to have different types of outputs (and also for developers to create a module using an existing format) extends the use of these natural language outputs to much more than simple printed monographs but also, for example, online publishing with automatic page setting for example with a predefined CSS style sheet. Thus, thanks to the dependencies to other CDM objects and functions, the users have a pretty vast number of possibilities to create, use and publish natural language descriptions.

Generating simple keys is now possible with the CDM library, it is an entirely automatic process that takes as input a CDM database, which means that after having persisted his data, a taxonomist can directly use this functionality without any extra work. Furthermore, it is up to him to combine the different CDM tools in order to meet his needs and, for example, print diagnoses* in natural language from the paths leading to different taxa. Of course, plenty of improvements could be made in the future, because taxa have specific characteristics that imply different ways to create the most useful key; the “target public” is also to be taken into account for it will influence the complexity of the key, and so on. For example, different possibilities to weigh the features could be considered.

In conclusion, these new functionalities are a pretty good example of how the CDM, by a good interconnection between its components, facilitates and extends the work of the scientist.

*minimal specific description