Writing bibliographic tools with *pybliographer*

Frédéric Gobry

February 2, 2006

Contents

Chapter 1

Introduction

pybliographer is a developer-oriented framework for manipulating bibliographic data. It is written in *python*[1], and uses extensively the dynamic nature of this language.

pybliographer does not try to define another standard format for bibliographic data, nor does it solely rely on a single existing standards. Standards are important in order to allow for interoperability and durability. Unfortunately, real-world data often contain a great number of mistakes, or reflect certain local conventions. *pybliographer* is on the *pragmatic* side of considering these issues as part of its business: most of the parsing tasks can be easily overriden and specialized in order to *fit the code to the data*, and not the other way around.

1.1 Basic concepts

pybliographer deals with sets of `Record`s, stored in a so-called `Database`. This database can be actually implemented on top of different systems. Two are available today, one based on a single XML file, using a custom XML dialect, the other based on Berkeley DB[2], a very efficient database system.

Each record represents an elementary object you want to describe, and has a number of *attributes*. For instance, if you are describing a book, one attribute will be its *title*, another its *ISBN*, etc. Each of these attributes can contain one or more values, all of the same *type*. To continue the description of our book, we probably have the *author* attribute, which contains as many `Person` values as there are authors for the book. All the values of a given attribute are of the same type.

In some cases, simply having this flat key/value model to describe an object is not enough. *pybliographer* allows, for every value of every attribute, to provide a set of *qualifiers*. These qualifiers are also attributes which can hold one or more values. If my book, or information about the book, is available via the internet, I can provide a *link* attribute, but for each of the actual `URL`s provided, I might wish to add a *description* qualifier, which will indicate, say, if the URL points to the editor's website, or to a review, etc.

This nesting of objects is best described in figure 1.1.

[1] see `http://python.org/`
[2] see `http://www.sleepycat.com/`

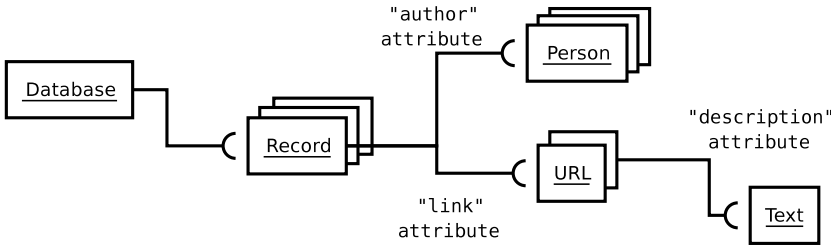

Figure 1.1: Objects manipulated in *pybliographer*

pybliographer comes with a set of defined attribute types, like `Person`, `Text`, `Date`, `ID` (see the `Pyblio.Attribute` module for a complete list), and can be extended to support your own types.

1.1.1 The database schema

Even though attributes are typed, the data model described above is quite flexible. In order for *pybliographer* to help you checking that your records are properly typed, it needs to know the database schema you are using. This schema, usually stored in an XML file with the extension `.sip`, simply lists the known attributes with their type and the qualifiers it allows for its values. Some `.sip` files are distributed with *pybliographer*, and can be seen in the `Pyblio.RIP` directory.

In addition to validation information, the schema contains human-readable description of the different fields, possibly in several languages, so that it can be automatically extracted by user interfaces to provide up-to-date information.

1.1.2 Taxonomies

TODO

1.1.3 Result sets

TODO

1.2 Manipulating data

TODO

1.2.1 Sorting

TODO

1.2.2 Searching

TODO

1.3 Importing and exporting

TODO

1.4 Citation formatting

TODO

Chapter 2

Extending *pybliographer*

TODO

2.1 Specializing a parser

TODO