Going open source: some lessons learned from the development of OpenRecLink

Rumo ao software aberto: algumas lições aprendidas com o desenvolvimento do OpenRecLink

Rumbo al código abierto: algunas lecciones aprendidas del desarrollo de OpenRecLink

Kenneth Rochel de Camargo Jr. [1]
Claudia Medina Coeli [2]

[1] *Centro Biomédico, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, Brasil.*
[2] *Instituto de Estudos em Saúde Coletiva, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil.*

Correspondence
K. R. Camargo Jr.
Departamento de Planejamento e Administração em Saúde, Centro Biomédico, Universidade do Estado do Rio de Janeiro.
Rua São Francisco Xavier 524, 7º andar, Bloco D, Rio de Janeiro, RJ 20559-900, Brasil.
kenneth@uerj.br

Abstract

Record linkage is the process of identifying and merging records across different databases belonging to the same entity. The health sector is one of the pioneering areas of record linkage techniques applications. In 1998 we began the development of a software package, called RecLink that implemented probabilistic record linkage techniques. In this article we report the development of a new, open-source version of that program, now named OpenRecLink. The aim of this article is to present the main characteristics of the new version and some of the lessons learned during its development. The new version is a total rewrite of the program, based on three goals: (1) to migrate to a free and open source software (FOSS) platform; (2) to implement a multiplatform version; (3) to implement the support for internationalization. We describe the tools that we adopted, the process of development and some of the problems encountered.

Software; Database; Record Linkage

Resumo

Linkage *de registros é o processo de identificação e fusão de registros entre diferentes bases de dados pertencentes à mesma entidade. O setor de saúde é uma das áreas pioneiras na aplicação de técnicas de* record linkage*. Em 1998, iniciamos o desenvolvimento de um programa chamado RecLink, que implementava as técnicas de relacionamento probabilístico. Neste artigo, relatamos o desenvolvimento de uma nova versão de fonte aberta desse programa, agora chamado OpenRecLink. O objetivo deste artigo é apresentar as principais características da nova versão e algumas das lições aprendidas durante o seu desenvolvimento. A nova versão é uma reescrita total do programa, com base em três objetivos: (1) migrar para uma plataforma de software livre e de código aberto, (2) implementar uma versão multiplataforma e (3) implementar o suporte para a internacionalização. Descrevemos as ferramentas adotadas, o processo de desenvolvimento e alguns dos problemas encontrados no seu desenvolvimento.*

Software; Base de Dados; Relacionamento de Dados

Introduction

Record linkage is the process of identifying and merging records across different databases belonging to the same entity [1]. As a result, it creates a new database that has more variables, improving the ability in answering research questions. It can also be used to identify records that refer to the same entity within a single database [1], which is useful for both data quality improvement and the study of repeated health events.

The health sector is one of the pioneering areas of record linkage techniques applications. Although new techniques have been developed and applied, Winkler [2] pointed out that none of them has consistently outperformed the probabilistic linkage techniques based on the Fellegi-Sunter model.

In 1998 we began to develop an application for probabilistic record linkage, out of the need to have an affordable tool that would implement Felegi and Sunter's methodology of probabilistic record linkage [3]. Right from the start we intended to have the program developed as a Free and Open Source Software (FOSS), for a number of reasons: it is more appropriate to the scientific ethos; it can attract developers beyond one specific team, thus speeding up the software lifecycle and, at least in theory, produce stronger code; and finally, it can assure the survival of the program itself, even if the original authors choose not to continue its development [4]. We chose to use the C++ language due to its power, general availability and support. We chose a programming tool that offered support for graphic user interface development under 32 bit Windows (Microsoft Corp., USA). We used a third-party vendor library for the database back-end, which supported the venerable xbase (dbf) file format. Although we distributed freely the package from the beginning, we could not open-source it, because it relied on third-party code to which we had no rights to distribute (and were in fact forbidden to do so). The software we developed – called RecLink – went through a number of iterations and reached a stable level in version 3 [5,6], which became widely used in Brazil [7].

With time, however, some of the problems of the approach we adopted became apparent. There were issues with index files when the number of records was too large – and that became increasingly common in the experience of our users. The dbf format itself had some limitations that were increasingly becoming apparent (especially the maximum number of variables). The graphic interface framework depended on a proprietary implementation of another language, not C or C++, which locked us to that specific vendor. These problems became clearer on two occasions.

First, a collaborator developed a routine to estimate linkage parameters based on the E/M algorithm [8], but we realized we would not be able to distribute it, since it included a specific library licensed under the Gnu Public License (GPL), which determines that all code should be made available to end users (unlike the Less General Public License, LGPL, that allows for some exceptions), something which we could not do. That part of the library had to be rewritten in order to allow the distribution of the program. Second, somewhere during the development of the third version of the program, the software company that produced the C++ development tools we were using signalized their discontinuation; although that decision was reversed, it made us deeply aware of the risks of vendor lock-in in terms of software development.

At the same time, we started adopting Linux as our operating system (OS) of choice (more specifically, the Ubuntu distribution), and decided to have a version of the software available that would run under that OS as well, something that was not possible with the toolset we were using. It became clear that a new version was necessary. We had three design goals in mind: making the software available in different OSs, at first Windows and Linux; making the software available in different languages, initially Portuguese and English; and improving the overall performance of the system.

The aim of this article is to present the main characteristics of the new version and some of the lessons learned during its development.

Characteristics of the program

We tried to keep the user interface as similar as possible to the previous version, adopting the name "OpenRecLink" to both signalize continuity and the new open source nature of the program. As in the previous version, most operations require two steps: the definition of a configuration file, using a graphic interface ("assistant") through a step-by-step procedure which includes checks for errors in the configuration, and then batch processing the data files based on the generated configuration. A major modification, compared to the previous version, was in the deduplication routine, which was totally redesigned. The main changes in this routine were a better display of possible duplicate records and an easy mechanism to change unique identifiers, in addition to marked performance enhancements.

Another area that underwent major changes was the file format adopted (see technical details in the following sections). Because of the new format, routines for file management (creation, indexing, copying, exporting and importing) were added to the program.

The general operation of the program is demonstrated by its menu structure, shown with comments in Figure 1.

Choice of development tools

After some research, and having decided to stick to C++ as our language of choice, we decided to use wxWidgets (http://www.wxwidgets.org) as our graphic framework of choice, using the gnu toolchain (the gcc compiler suite), and codeblocks (http://www.codeblocks.org/), an IDE built with wxWidgets and with several facilities for developing code with that framework, including a graphic screen designer, called wxSmith.

The gcc toolchain is available in practically every major OS in use, is one of the most standards-compliant toolsets available, and produces high quality binary code. Under Windows we also use a command-line environment called MSYS (http://sourceforge.net/projects/mingw/files/MSYS/), that provides a very similar set of tools to those available under Linux, thus allowing us to use a single set of makefiles to create versions for both systems, which considerably accelerated the process of generating new versions for distribution.

Although all the tools used in the development of the new version allow the development of closed-source commercial programs, we decided to use an open-source model, adopting the GPL version 3 for licensing. WxWidgets not only provides multi-platform support, but also includes support for the gettext framework, which allows multi-language support without recompilation; a tool scans the C++ code and identifies strings that were marked using a specific convention, and generates a text file with all such strings (a .mo file), which can then be translated into different languages and compiled into as many .po files as necessary, which can be selected at runtime by the program.

The development process

Development started by the end of 2009, being hosted at Sourceforge (http://reclink.sourceforge.net/). For a first iteration, we still used the dbf file format. We designed a C++ class wrapper that encapsulated the library we were using (also free and open source), in order to make it easy to transition to other database back-ends in the future, a decision that later proved its value.

We made a few preliminary alpha versions of the software and started researching alternatives for the dbf file format. We ruled out from the start any full-fledged DBMS from consideration; previous attempts, still in the closed-source phases of development, showed that all the overhead introduced by SQL interpretation layers, journaling and other features introduced an undesirable level of complexity and performance problems at the same time.

We adopted a dual-license library (free for open-source programs, and royalty-based for closed-source ones) at first, that seemed to have the simplicity and performance that we wanted. Although the former was true, when we finally managed to do actual load tests with the new version of the program, we were dismayed to find that a test linkage had a tenfold increase in processing time, even when comparing an old 32 bit computer with a single-core processor with a modern, 64 bit, double core one.

All the indexing systems we used up to this point were based on B-Tree variants. Although perfectly suited for interactive databases, that are being queried and updated all the time, they are not necessarily suited for the kind of batch processing that takes place in our program. As data files grow in size, node and leaf splitting and recombining become more frequent, and that takes a toll on the overall performance.

We decided to take a brute-force approach, creating an indexing system based on a two step process: first, the data file is read from beginning to end, generating a second file which contains as many records, each consisting of the index key (a fixed size character sequence) and a pointer to the record in the data file. Then the generated index file is sorted in-place, using an adapted version of the qsort algorithm. Searches on the index file are made with a simple binary search, with one important adaptation: since we admit multiple entries with the same key, when a key is found it is not necessarily the first occurrence in the index file, so the program skips back in the index file to find the first occurrence, since this is necessary for the correct operation of many routines in the program.

We had to create our own record file format, loosely based on the dbf format, but far less restrictive. Field names can be up to 15 characters, the number of fields can be anything expressed by an integer (usually a fairly large number, regardless of implementation), any field can be up to 1k (1024) bytes in length, and the total size of a record can be up to 32k bytes in length.

Figure 1

OpenRecLink menu structure and main functions.

File: *Set of tools to work with OpenRecLink data files and indexes, including exporting/importing data.*
Submenus:
- Tools
  - File
    - Visualize
    - Structure
    - Reset
    - Index
    - Quit
  - Move
    - Annex
    - Export
    - Import
    - Import dbf
- Activity log
- About
- Quit

Copy: *Routines for selective copy of parts (records and/or fields) of data files.*
Submenus:
- Configure
- Run

Standardize: *Routines for standardizing data.*
Submenus:
- Configure
- Run

Linkage: *Routines for blocking and comparison of links, including estimating m and u weights.*
Submenus:
- Configure
- Generate frequency
- Analyse frequency
- Generate matrixes
- Calculate
- Run
- Reduce pairs
- Mark/unmark
- SQL export

Deduplication: *Routines for identifying duplicate records on data files.*
Submenus:
- Configure
- Run
- Inspect

Combine: *Routines for automated classification, clerical review and combining linked files.*
Submenus:
- Examine
- Run

Join:
Submenus: *Routines for linkage based on a shared unique identifier.*
- Configure
- Run

Having created the wrapper class previously all those transitions between different data backends were relatively easy to undertake. We included an information string in the file header to make sure that it is indeed the correct type of file that is being opened, and also to allow differentiation of the 32 and 64 bits of the data and index files, since the aforementioned differences in integer type sizes make the two incompatible with each other.

We were still worried about performance issues with the adopted strategy, but tests made so far indicate that there were definite gains. A test program made specifically to compare indexing strategies, comparing our solution to a B-Tree implementation [9] originally made in C, which we adapted as a C++ class, showed a significant gain: generating an index with a 50-byte key size for a file with 1,529,952 records took 12:38 minutes/seconds with the B-Tree (best result after tweaking certain parameters, especially page size) and 5:23 minutes/seconds for the sorted index (2:35 min for generating the file and 2:48 min for sorting). Using the generated indexes to create a sorted list of records, output to a text file, took similar amounts of time (2:56 min and 2:44 min, respectively). The tests were performed on a computer with an Intel Core i7 computer with 32 Gbytes of RAM, a 1 Terabyte hard disk, running Ubuntu Linux 14.04, 64 bits. Obviously larger files will lead to different results, and the increase in processing time is not linear in either case, but nevertheless this comparison is relevant.

During the early testing phases of the program we encountered a specific performance problem with the indexing scheme we adopted during the clerical review process, which required a workaround; explaining that requires a closer look at the operation of two modules of the program. For the linkage operation itself, as explained before the program will take a previously created configuration file, and will read one of the files from beginning to end, comparing each record to those within a zone in the other file (the block, specified by user-defined parameters), and generating a score computed for all the determined field comparisons in the configuration; for each comparison a record is generated in another file, called the pairs file, containing the final score and the pointers to the compared records in each file (low values can be kept from generating records in the pairs file, according to a user-defined threshold).

Another module of the program allows a kind of "virtual joining" of the linkage files, looking into each record of the pairs file and displaying the data of the corresponding records in the two linked files. This is made to allow for a clerical review of the linkage process. Navigation in that module is made according to blocks with the same score. Since multiple pairs can have the same score, that created a problem for the search function. Going back to the point of the index file corresponding to the beginning of a block that has tens of millions of records is very time consuming. In order to get around that, that specific routine was changed to create a sort of virtual index of the index, registering where each block begins.

Releasing new versions

We have followed one of the principles of FOSS developments, to release new versions as often as possible [4,10]. Before releasing new versions, we perform tests of performance and functionality with two test databases that we created for this purpose from actual data, with known true matches and true no matches. As new versions are released, they are being deployed in our research as beta releases, with further testing being done with actual use. Both Linux and Windows 64 bit versions are tested at this stage. Early user feedback drives further development.

Discussion

Whereas in the articles that we previously published dealing with the closed source version of the program [5,6] our goal was to present the operating aspects of the software, in this text one of our concerns, following Prlić & Procter [10] is to promote the project and try to create, or in this case expand, a community around it, hopefully involving more developers. From our experience, we believe that there are already enough researchers in public health who can participate in the development of open source programs.

As for the lessons learned, first of all adopting good programming practices is essential. The separation between business logic and user interface, that was already adopted in the closed version, proved fundamental in the transition. Although the code was rewritten almost in its entirety, in order to remove dependencies on closed source third party libraries, the classes that were designed and the general approach to the problems were maintained and served as a blueprint for the new version.

Second, the tools we chose proved to be adequate to the task. The general availability of many multi-purpose open source libraries in C and C++, including the GUI framework itself, was vital for the successful development of the new

version. Although we would rather concentrate on the Linux version, we have a user base that only works with Windows, and the multiplatform capabilities of wxWidgets were one of the key features for its adoption.

This, however, introduced another level of complexity, particularly in terms of testing; one of the principles proposed by Prlić & Procter [10], 'be your own user', is harder to adhere to when it comes to using different platforms. We can count on our existing user base to help us in this regard.

In parallel to the more technical aspects of coding, we have invested in training different groups in using the software, which has proven to be a key aspect in the diffusion of its use. Although we do have a website for the program, currently our main channel of interaction with users is via e-mail.

Finally, we have striven to create adequate documentation for users, not so much for developers. Limitations in resources have prevented us from fully addressing this issue; that, and the improvement of communication and interaction with users and potential developers, are areas in which we intend to invest additional efforts.

Once again referring back to Prlić & Procter [10], we should not lose sight of the fact that the software we develop is an instrument for our scientific research, and not an end in itself.

Resumen

La vinculación de registros es el proceso de identificar y combinar registros a través de diferentes bases de datos que pertenecen a la misma entidad. El sector de la salud es una de las zonas pioneras en el récord de técnicas de vinculación de aplicaciones. En 1998 se inició el desarrollo de un paquete de software, llamado RecLink, que implementaba técnicas de registro de vinculación probabilísticos. En este artículo se presenta el desarrollo de una nueva versión de código abierto de ese programa, ahora llamado OpenRecLink. El objetivo de este artículo es presentar las principales características de la nueva versión y algunas de las lecciones aprendidas durante su desarrollo. La nueva versión es una reescritura total del programa, sobre la base de tres objetivos: (1) migrar a un software libre y de código abierto de la plataforma (FOSS), (2) implementar una versión multiplataforma; (3) poner en práctica el apoyo a la internacionalización. Se describen las herramientas que hemos adoptado, el proceso de desarrollo y algunos de los problemas encontrados.

Programas Informáticos; Base de Datos; Vinculación de Datos

Contributors

K. R. Camargo Jr. and C. M. Coeli were responsible for elaboration, revision and approval of the final version of the article.

Acknowledgments

The study received funding from CNPq and FAPERJ. C. M. Coeli and K. R. Camargo Jr. received research fellowship grants from CNPq.

References

1. Christen P, editor. Data matching concepts and techniques for record linkage, entity resolution, and duplicate detection. Berlin/New York: Springer; 2012.
2. Winkler WE. Foreword. In: Christen P, editor. Data matching concepts and techniques for record linkage, entity resolution, and duplicate detection. Berlin/New York: Springer; 2012. p. vii-viii.
3. Fellegi IP, Sunter AB. A theory for record linkage. J Am Stat Assoc 1969; 64:1183-210.
4. Raymond ES. The cathedral and the bazaar. First Monday 1998; 3(3). http://ojphi.org/ojs/index.php/fm/article/view/578/499 (accessed on 22/Jun/2014).
5. Camargo Jr. KR, Coeli CM. *Reclink*: aplicativo para o relacionamento de base de dados, implementando o método *probabilistic record linkage*. Cad Saúde Pública 2000; 16:439-47.
6. Camargo Junior K, Coeli C. RecLink 3: nova versão do programa que implementa a técnica de associação probabilística de registros (probabilistic record linkage). Cad Saúde Colet (Rio J.) 2006; 14:399-404.
7. Silva JPL, Travassos C, Vasconcellos MM, Campos LM. Revisão sistemática sobre encadeamento ou linkage de bases de dados secundários para uso em pesquisa em saúde no Brasil. Cad Saúde Colet (Rio J.) 2006; 14:197-224.
8. Junger WL. Estimação de parâmetros em relacionamento probabilístico de banco de dados: uma aplicação do algoritmo EM para o Reclink. Cad Saúde Colet (Rio J.) 2006; 14:225-32.
9. Stevens A. C database development. 2nd Ed. New York: MIS Press; 1992.
10. Prlić A, Procter JB. Ten simple rules for the open development of scientific software. PLoS Comput Biol 2012; 8:e1002802.

Submitted on 16/Mar/2014
Final version resubmitted on 28/Jul/2014
Approved on 30/Jul/2014