ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

SoftwareX ▮ (▮▮▮▮) ▮▮▮–▮▮▮

SoftwareX

www.elsevier.com/locate/softx

PiCO QL: A software library for runtime interactive queries on program data

Marios Fragkoulis*, Diomidis Spinellis, Panos Louridas

Athens University of Economics and Business, Patision 76, Athens, Greece

Received 1 December 2015; received in revised form 19 July 2016; accepted 20 July 2016

Abstract

PiCO QL is an open source C/C++ software whose scientific scope is real-time interactive analysis of in-memory data through SQL queries. It exposes a relational view of a system's or application's data structures, which is queryable through SQL. While the application or system is executing, users can input queries through a web-based interface or issue web service requests. Queries execute on the live data structures through the respective relational views. PiCO QL makes a good candidate for ad-hoc data analysis in applications and for diagnostics in systems settings. Applications of PiCO QL include the Linux kernel, the Valgrind instrumentation framework, a GIS application, a virtual real-time observatory of stellar objects, and a source code analyser.

Keywords: SQL; Main memory data structures; Data analysis; Program data

Code metadata

Current code version	v11
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00086
Legal Code License	Apache 2.0
Code versioning system used	Git
Software code languages, tools, and services used	C/C++, Ruby
Compilation requirements, operating environments & dependencies	GCC/G++, Ruby interpreter, Unix-like system: autotools
If available Link to developer documentation/manual	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00086/blob/PiCO_QL-application-release/README.mediawiki
Support email for questions	mfg@aueb.gr

Software metadata

Current software version	1.1
Permanent link to executables of this version	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00086
Legal Software License	Apache 2.0
Computing platform/Operating System	Linux, OS X, Unix-like, web based
Installation requirements & dependencies	Ruby, SQLite3, SWILL, Boost (optional)
If available, link to user manual — if formally published include a reference to the publication in the reference list	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00086/blob/PiCO_QL-application-release/README.mediawiki
Support email for questions	mfg@aueb.gr

1. Motivation and problem description

Many software applications depend on efficient data processing and analysis. Whether it is GPS data produced by hand-held devices or astronomy data produced by enhanced

* Corresponding author.
E-mail address: mfg@aueb.gr (M. Fragkoulis).

http://dx.doi.org/10.1016/j.softx.2016.07.002
2352-7110/© 2016 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

telescopes, the availability of more data is pushing the bar of data processing requirements higher.

A common solution to this challenge is to keep data in memory for efficient processing. Advances in memory hardware capacity help this situation: 8 GB of memory is common in personal laptops, 16 GB is very feasible, and sophisticated server machines exist with 125 GB. On the software end, general purpose programming languages provide (a) main memory data structures that can express sophisticated relationships and (b) efficient algorithms to apply operations on them.

What many general purpose programming languages are missing however is an expressive query language for sophisticated data analysis and an interpreter to shorten the query life cycle towards a data base interface experience. Database management systems (DBMS), on the other hand, require an expensive model transformation and unnecessary dependencies when it only comes to querying program data.

We propose another path where it is possible to write and execute interactive ad-hoc queries on program data structures at runtime. Queries are input in a high-level language suitable for data analysis through a web interactive interface or web-service and executed in place on the application's data structures. This is PiCO QL, an SQL relational interface for querying C/C++ program objects. Notably, PiCO QL does not impose any transformation to the native object model, which falls under the object-relational mapping problem [1]; it only provides a queryable relational view on top of it.

2. Use preparation

Prior to use with an application, PiCO QL requires a one-off setup process that includes three tasks:

1. Register each data structure with PiCO QL using a simple method call within the application's code and start the query library using another method call.
2. Write a relational representation of the selected data structures in PiCO QL's domain specific language (DSL), which resembles relational table definitions. A detailed description of the relational representation and the DSL is available in Ref. [2]. A number of applied examples are present in PiCO QL's codebase[1] and the project's wiki pages highlight the installation and application plugin process.[2]
3. Compile the application together with PiCO QL by linking to the PiCO QL library.

Fig. 1 depicts the steps of the setup process.

3. Software architecture

Fig. 2 depicts PiCO QL's software architecture. It consists of: the PiCO QL API as described in the previous section, the SQLite [3] database query engine, an implementation of SQLite's virtual table API, a source-code compiler or generator, and a web interface module.

SQLite is an embeddable open source relational database system that supports SQL ANSI92. It features a virtual table module, which provides users with the building blocks for attaching arbitrary data sources to SQLite's query engine. Data sources can benefit from the relational facet by implementing SQLite's virtual table API. Because SQLite is embeddable, it is appropriate for use as a query interface plugin to applications.

PiCO QL implements SQLite's virtual table API in order to present a relational query interface to object-oriented program data. The virtual table API implementation splits in two parts. The one part concerns API methods fixed for all applications, such as the ones that open or close a virtual table. The other part implements the API methods specific to each application. These carry out query processing operations, such as searching a virtual table or returning a column of it. Because virtual tables mirror program data structures, the query processing methods have to be generated according to the data structures at hand.

The source code generator or meta-programming module, which is implemented in Ruby, takes a specification of virtual tables linked to an application's data structures and generates the implementation of the query processing methods. The virtual table specification, which is written in the PiCO QL DSL, resembles relational table definitions.

The web interface module uses the SWILL [6] library for passing query input from the web interface, which is presented, e.g., at port 8080 of localhost, to the SQLite engine. The interface hosts the schema of the relational representation to facilitate query input.

4. Applications of PiCO QL

We have used PiCO QL both in system settings and application domains. Within systems, PiCO QL serves as an ad-hoc diagnostic tool with a high-level programming language. We tested its capabilities within the Linux kernel and the Valgrind instrumentation framework. We also embedded PiCO QL in three applications of different domains [2]. QLandKarte is a GIS application that visualises GPS data, such as bike or running routes. The second one, Stellarium, is a virtual real time observatory of stellar objects. Finally, CScout [4] is a source code analyser and refactoring browser for collections of C applications. Three queries, one for each application, are presented in the last three rows of Table 1.

The application of PiCO QL [5] in the Linux kernel is interesting from a number of standpoints. A wealth of kernel data structures is in our disposal, such as the list of processes, files, and devices in the system. The challenge is to combine data structures together in queries in order to turn data into information. Performance diagnostics fits well in this scenario. We performed queries that brought together the network, memory, file, and process subsystems, such as the one described in the second row of Table 1. Security is another application area we have touched by firing queries to identify reported system vulnerabilities (see the first row of Table 1). These and other queries are presented in detail in Ref. [5] together with performance metrics.

[1] https://github.com/mfragkoulis/PiCO_QL/tree/ (a) PiCO_QL-application-release/examples (b) PiCO_QL-kernel-release/src/Linux-kernel-mod (c) PiCO_QL-Valgrind-release/src/Valgrind-mod.

[2] https://github.com/mfragkoulis/PiCO_QL/wiki/Quickstart.