Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 292–307
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

An infrastructure to support interoperability in reverse engineering

Nicholas A. Kraft [a], Brian A. Malloy [a,*], James F. Power [b]

[a] Department of Computer Science, Clemson University, Clemson, SC, USA
[b] Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland

Received 6 July 2006; accepted 25 October 2006
Available online 4 January 2007

Abstract

The reverse engineering community has recognized the importance of interoperability, the cooperation of two or more systems to enable the exchange and utilization of data, and has noted that the current lack of interoperability is a contributing factor to the lack of adoption of available infrastructures. To address the problems of interoperability and reproducing previous results, we present an infrastructure that supports interoperability among reverse engineering tools and applications. We present the design of our infrastructure, including the hierarchy of schemas that captures the interactions among graph structures. We also develop and utilize our implementation, which is designed using a GXL-based pipe-filter architecture, to perform a case study that demonstrates the feasibility of our infrastructure.

Keywords: Reverse engineering; Data interoperability; Graph-based tools; Tool interoperability; GXL

1. Introduction

In reverse engineering, interoperability is the cooperation of two or more systems to enable the exchange and utilization of data [27]. The reverse engineering community has recognized the importance of interoperability among tools [9], as well as the difficulty in facilitating interoperability among these tools [2,3,6,12,19,34,43]. In their roadmap for reverse engineering, Müller et al. identify the lack of adoption of infrastructures as one of the biggest challenges to increasing the interoperability of data, so that the effectiveness of reverse engineering approaches hinges on addressing this challenge [44]. An infrastructure is a set of interconnected structural elements, such as tools and schemas, that provide a framework for supporting features and facilities, such as interoperability and reuse. The lack of interoperability among reverse engineering tools and other software utilities has been highlighted as a contributory factor to the lack of adoption of available infrastructures [44].

The issues involved in promoting interoperability among reverse engineering tools and applications have been discussed at the Dagstuhl Seminar on *Interoperability of Reengineering Tools* [7]. At the seminar, the participants identified three levels at which interoperability should be applied: low-level syntax, middle-level graph structures, and high-level architectures. The importance of facilitating interoperability is becoming increasingly recognized for its importance in permitting reuse of reverse engineering artifacts, as well as enabling the reproduction of results from previous scientific research.

The two important activities involved in most research endeavors entail the development of an approach that is an improvement on existing approaches and then conducting experiments to show that the new approach is an improvement over existing approaches. To evaluate the new approach, the researcher is typically required to implement at least one previously developed technique as an unbiased basis for comparison with the newly developed technique. However, even after the previously developed technique is implemented, the researcher is frequently unsure of the

* Corresponding author.
E-mail addresses: nkraft@cs.clemson.edu (N.A. Kraft), malloy@cs.clemson.edu (B.A. Malloy), jpower@cs.nuim.ie (J.F. Power).

0950-5849/$ - see front matter © 2007 Published by Elsevier B.V.
doi:10.1016/j.infsof.2006.10.014

correctness of the implementation or the correctness of the generated results. Thus, comparison of competing approaches is difficult and all too frequently impossible. For example, researchers in language design and implementation have reported considerable difficulty in replicating results in generating call graphs and points-to-analysis, even for C programs [5,46].

To address the problems of interoperability and reproducing previous results, we presented an infrastructure that supports interoperability among reverse engineering tools and applications [35]. In this paper, we expand on the infrastructure in several important directions. We develop and extend a schema hierarchy that is central to our approach, detail the interactions among instances of the schemas, and illustrate several of the schemas. In addition, we describe our implementation of the essential components of the infrastructure, including a linking process for unifying instances of an API for all translation units in a C++ program, and present the results of a case study for our infrastructure that includes ten popular open source applications and libraries as a test suite. As part of our study, we apply XSLT style sheets to GXL instance graphs. Our implementation of the infrastructure, as well as GXL versions of the schemas in the hierarchy and our XSLT style sheets, are available in our web repository [47].

The contribution of our work is the design and implementation of our infrastructure to support interoperability, as well as a case study that demonstrates the feasibility of our infrastructure. The design of our infrastructure includes a schema hierarchy and details of the interactions among the schemas. The implementation of our infrastructure includes a collection of tools that communicate via a GXL-based pipe-filter architecture. In particular, we provide an application programmers interface (API) and a set of tools that leverage the API to perform reverse engineering tasks, including: construction of graphical program representations [35,33], computation of metrics [28], and static analysis [21]. Thus, our infrastructure operates at levels one and two as specified by Sim [9].

In Section 2 we describe previous research that relates to our infrastructure. In Section 3 we provide details about the infrastructure, including our hierarchy of canonical schemas. In Section 4 we present g^{4}re, our implementation of the infrastructure, and describe the instantiation and linking processes for g4api. In Section 5 we list results of a case study that investigates the feasibility of our infrastructure using 10 open-source applications and libraries as a test suite. Finally, in Section 6 we draw conclusions and describe future work.

2. Related work

In this section we describe the work that relates to the design and implementation of our infrastructure. In particular, we describe research on infrastructures for reverse engineering, evaluating reverse engineering tools, and linking in reverse engineering tools.

2.1. Infrastructures for reverse engineering

One of the earliest approaches to providing a general framework for interoperability is the ECMA Reference Model, the "Toaster Model", which outlines the functionality required to support a tool integration process [48]. The dimensions of functionality addressed by the model include: data integration, provided by the repository manager; control integration, provided by the subsystem interaction manager; presentation integration, provided by the user interaction manager; and process integration, provided by the development manager.

One of the earliest approaches to a reverse engineering infrastructure is the LSME system by Murphy and Notkin [45]. This system is based on lexical analysis and specifically identifies the ability to add additional source languages and extractors as central to the approach. This flexibility is demonstrated by applying the approach to extracting source models for ANSI C, CLOS, Eiffel, Modula 3 and TCL.

Kullbach et al. present the EER/GRAL approach to graph-based conceptual modeling of multi-lingual systems [36]. In this approach, models to represent information from a single language are built and then integrated into a unified model. A graph query language is available to perform queries on the unified model.

Dali is a collection of various tools in the form of a workbench for collecting and manipulating architectural information [31]. The Dali workbench was designed to be *open*, so that new tools could be easily integrated, and *lightweight*, so that such integration would not unnecessarily impact unrelated parts of the workbench. Kazman et al. identify an extraction phase, encompassing both parsing and profiling, accumulating information in a repository, which then feeds visualization and analysis phases. They use an SQL database for primary model storage, but then use application specific file formats to facilitate interchange between tools.

The Dali architecture is echoed by Salah and Mancoridids in their *software comprehension environment*, which has a three-layer architecture composed of a data gathering subsystem, a repository subsystem, and an analysis and visualization subsystem [51]. Their environment supports both static and dynamic analysis of Java and C++ programs, and information can be accessed using either SQL or a specialized higher level query language.

Finnigan et al. describe a *Software Bookshelf*, that was originally designed to support converting PL/I source code to C++ [13]. Their information repository, describing the content of the bookshelf, is accessed through a web server using object-oriented database technology. An implementation of these ideas as the *Portable Bookshelf* (PBS) is based around a toolkit that includes a fact extractor, manipulator, and layout tools. This "pipeline philosophy" has since evolved into the SWAG Kit and the LDX/BFX pipeline, each emphasizing collections of stand-alone tools communicating only via well-defined inputs and outputs [22].