Towards an Integrated Toolset for Program Understanding†

John Mylopoulos Martin Stanley Kenny Wong Morris Bernstein
Renato De Mori Graham Ewart Kostas Kontogiannis Ettore Merlo
Hausi Müller Scott Tilley Marijana Tomic

Abstract

This paper describes some early results of a three-year project to develop an integrated toolset for program understanding. The implemented integration architecture involves both a global repository for all tools serviced by the architecture and a software bus serving communications among tools.

1 Introduction

Program understanding is the process of developing mental models of a software system's intended architecture, purpose, and behaviour. Software engineers spend more time understanding existing code than they do designing, programming, testing, or debugging. Moreover, the need for program understanding is growing in parallel with the amount of legacy code currently in use. Not surprisingly, there have been numerous research efforts to develop tools that provide assistance during the understanding process. These tools adopt a number of different approaches, including visualization, pattern-matching, and knowledge-based techniques. Despite successful results from each of these approaches, it is clear that no one approach or tool is sufficient by itself, and that the software engineer can best be served through a collection of tools that complement each other in functionality. This observation raises three research issues. First, what sort of an architecture can best support an integrated set of tools for program understanding? Second, what are the concrete benefits of this integration? Third, how well does an integrated toolset support a software engineer?

This paper reports on early results of a three-year project, called RevEngE, whose objectives include the development of an integrated toolset for program understanding. It offers tools for program visualization, system identification and discovery, and supports appropriate program understanding processes. The results to date include a prototype implementation of an integrated environment and initial experimentations with legacy code. RevEngE is part of an ongoing project on program understanding, based at the IBM Centre for Advanced Studies (CAS) [5]. The primary goal of this larger project [6] has been to apply program understanding technologies to improve the quality of software systems.

In this project, the SQL/DS (Structured Query Language/Data System) product is used as the testbed for all technologies under review. SQL/DS is a relational database management system, developed by IBM in the 1970s, operating on IBM's System/370 family of computers using either VM or VSE, and serving a large customer base.[1] SQL/DS is successful and mature, but is also evolving to run on new environments and support a growing functionality. For these reasons, it was deemed to be an excellent example of a legacy system.

The SQL/DS system consists of over one thousand compilable units containing roughly three million lines of source code written in PL/AS (Programming Language/Advanced Systems), an internal IBM language. PL/AS

†This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, the IBM Software Solutions Toronto Laboratory Centre for Advanced Studies, the Information Technology Research Centre of Ontario, the University of Victoria, and McGill University.

[1]SQL/DS, System/370, VM, VSE, and IBM are trademarks of International Business Machines Corporation.

is similar in structure and syntax to PL/I, but has many extensions that make it suitable as a "deep systems" language. For example, when necessary, the programmer can mix IBM System/370 Assembler code with PL/AS in a module.

Progress in the project has been accomplished through two vehicles. The first is the Software Refinery (also known as REFINE),[2] a commercial tool by Reasoning Systems Inc. that supports a parser for a variety of programming languages. The parser's output is a data structure representation of abstract syntax trees. Moreover, REFINE offers a rich language where users can define patterns to be matched against abstract syntax trees of the code under analysis. This tool was used extensively in earlier phases of the project in order to discover certain families of defects in the SQL/DS system. The second vehicle is the Rigi system, a research prototype developed at the University of Victoria to support program analysis and visualization. All tools under development in RevEngE are based on either REFINE or Rigi, or both.

Section 2 of this paper describes the integration architecture proposed for RevEngE, while Section 3 provides an overview of Rigi, recent improvements to its functionality, and its application to analyzing the structure of SQL/DS. Section 4 outlines extensions of REFINE to support pattern matching operations. Section 5 relates this work to other research efforts, and Section 6 summarizes the paper.

2 An Integration Architecture

The basic requirement of the overall architecture for the integrated toolset is to support both data and control integration for a given set of tools. Data integration is the sharing of data among a set of tools; control integration is the ability of tools to notify each other of events, and to activate other tools when needed. The design goals for the architecture are that it be open, modular, and capable of operating either

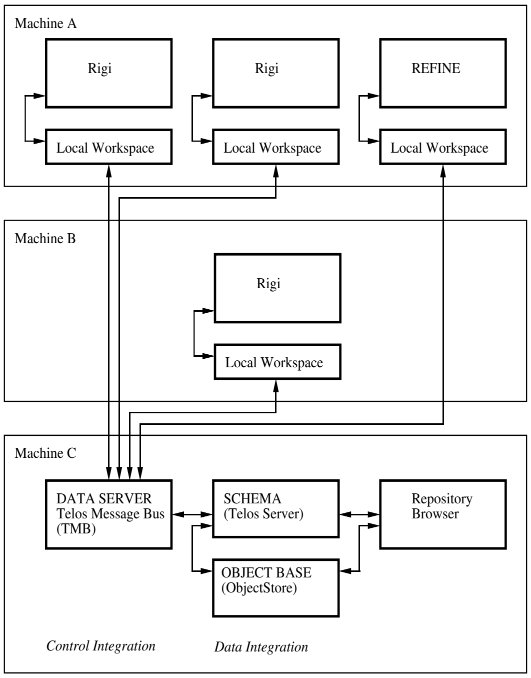

Figure 1: Control and data integration architecture

on a single host or over a network. An open system architecture was adopted because it makes the toolset served by the architecture extensible, while modularity facilitates customization (or even replacement) of components as needed. For instance, the underlying message transport software can be easily changed, if a more desirable system is found. It should also be relatively easy to add new tools.

The integration architecture of the system consists of two components. The first (labeled "Telos Server" in Figure 1) is responsible for data integration among tools, and is realized through a repository based on a single global schema that can accommodate all data handled by any one of the integrated tools. The second component (labeled "TMB") consists of a data server which is responsible for communicating information among tools either through the repository or directly (control integration).

The Telos server is implemented in C++ using ObjectStore as its persistent storage manager. The global schema is defined using an object-oriented information model (adopted from Telos [23]) which supports metaclasses,

[2]Software Refinery and REFINE are trademarks of Reasoning Systems Inc.

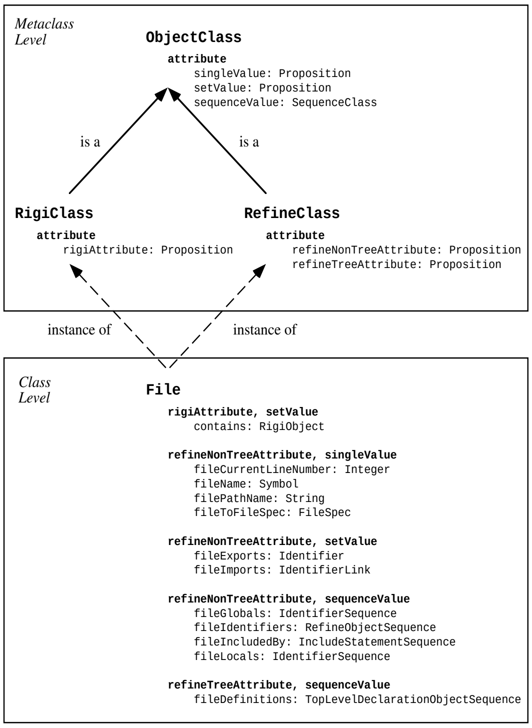

Figure 2: Portions of the Telos repository schema

multiple inheritance, and multiple instantiation. In particular, classes are objects and are instances of metaclasses. For example, an object `file007` may be an instance of the class `File` and `RigiModule`, reflecting that it represents a file and part of a Rigi module. Moreover, attributes are treated as objects and, therefore, are instances of attribute classes (which in turn are instances of attribute metaclasses, etc.).

Figure 2 shows a portion of the global schema that has been defined in order to integrate the tools Rigi and REFINE. At the metaclass level, the schema includes the metaclass `ObjectClass`, which has associated attribute metaclasses `singleValue`, `setValue`, `sequenceValue`. All instances of this metaclass can have associated attributes, which are classified under `singleValue`, `setValue`, `sequenceValue`, depending on the type of value they take. `ObjectClass` has two specializations, `RigiClass` and `RefineClass`. The former has as instances classes that are manipulated by the Rigi tool, while the latter has instances that are classes manipulated by the REFINE tool. As indicated in the figure, these metaclasses have attribute metaclasses that identify attributes used by Rigi and ones used by REFINE.

At the class level, the class `File` is defined as an instance of both `RigiClass` and `RefineClass`. This simply declares that a file object may be manipulated by both the Rigi and REFINE tools. `File` has many attributes classified under one or more attribute metaclasses. For instance, `fileCurrentLineNumber` is classified under `refineNonTreeAttribute` and `singleValue`, indicating that this attribute is single valued and that it is a non-tree attribute used by the REFINE tool. An instance of `File`, say `file007`, can now have associated attribute valued pairs that are classified under the attribute classes of `File`, including `fileCurrentLineNumber`, `contains`, etc.

The use of metaclasses for objects as well as attributes makes it possible to accommodate new tools as well as new types of information. The treatment of attributes as objects facilitates the partitioning of repository contents to suit the tools being accommodated and their users. For instance, a user of REFINE may want to load into the REFINE workspace all file objects with their associated REFINE attributes (but not their Rigi attributes). Expressing such a query in Telos is simple, precisely because attributes are objects.

The global schema that has been implemented so far can accommodate the two tools Rigi and REFINE used during program identification [3], where a given program is parsed and analyzed syntactically. It also accommodates them during design discovery, where various abstractions are discovered through pattern matching. An extension to the schema, planned for the second year of the project, will accommodate requirements specifications, process models, and domain knowledge, as well as the actual patterns used for their discovery.

The data server, called *Telos Message Bus* (TMB), is the kernel of the support system for tool interconnection and control integration. It is implemented in C++ using the software bus *MBus* [7] as the transportation mechanism. We chose MBus for the project, rather than use a commercial product, primarily because it gives us a small and simple system upon which we can build a message transport layer tailored specifically to the project's requirements. Its capabilities include reliable message transfer with selective broadcasting, and a simple message typing scheme.

The design goals of the TMB include extensibility, in the sense that a tool client must be able to dynamically inform the system that it can handle a new kind of request. For example, when REFINE implements a new complexity measure, it would register this capability so that other tools can make use of it. In addition, the TMB is designed to be extensible in the sense that it is a simple matter to add new tools to the architecture. A new tool would be added as another tool class to the currently existing classes for Rigi, REFINE, and the Telos Server. Instances of specific tools or users in the environment are uniquely identified by a combination of the login name, tool class, host ID, and process number.

Another important requirement is that the TMB supports point-to-point as well as broadcast communication. Tools can specify a particular domain of interest—the kinds of messages that it is interested in. A domain is a tuple consisting of a context (e.g., analysis update), a tool class, and a combination of login name, host ID, and process number. Domains also allow senders to specify particular receiver(s) of a message. The registry of contexts is expected to evolve over the course of the project.

Efficient transmission of bulk data is deemed to be critical, since that is the intended *modus operandi* for the integration architecture. That is, the repository is currently not optimized for queries of just single objects. Finally, the data server has been designed so that it lays the groundwork for an access control layer and a policy layer for implementing access control protocols and program understanding processes.

The TMB offers message passing using message objects, with client facilities for message creation, deletion, and archiving. Data is sent through a TMB message by creating a network object (a transportable representation of a Telos object) and sending it to another client on the TMB. For example, the REFINE tool might parse a program's source code, producing an abstract syntax tree representation of the program and send this tree via a set of network objects encapsulated in a TMB message to the Telos Server (simply another client on the TMB) for insertion into the repository. Then later on, the Rigi tool can request a certain subset of the objects previously created by REFINE and can get them through a TMB message containing the appropriate network objects from the Telos Server. Later on, the Rigi tool might request (using a TMB message) that the REFINE tool perform some additional analysis of the source code and that it report the results directly to Rigi. In this scenario, the REFINE tool sends a set of network objects in a TMB message to Rigi. Note that although the tools in this latter example are not using the repository at all, they are using the repository's global schema to communicate with each other.

3 The Rigi Tool

The basic function of the Rigi tool [18] is to discover abstractions from software representations and present them in a meaningful way to software engineers, thereby assisting their understanding of the subject system. Presentations of information in Rigi are graphical, and accommodate summarizing, querying, representing, visualizing, and evaluating system structure for large, evolving software systems.

Rigi encompasses a collection of research results integrated into a number of components, including a parsing subsystem, a distributed, multi-user workspace, and an interactive graph editor [21]. Rigi's representation for software structure is based on (k,2)-partite graphs [19], thereby making the analysis of program structure algorithmically tractable. In addition, [22] proposes a reverse engineering methodology, while [20] defines measures for evaluating the quality of structural abstractions. Also, [28] presents a documentation strategy using up-to-date views, and [29] offers an end-user programmable approach to extending the system's functionality.

The SQL/DS code was a useful testing ground for the Rigi approach, and led to a dramatic change to the underlying philosophy that a semi-automatic reverse engineering environment is better than a fully automatic one. This is because human cognitive abilities are still much more powerful and flexible than fixed algorithms. However, many of the operations performed during the initial decomposition of the SQL/DS code were repetitive and tedious. The analyst would still be in charge, but the process itself could be more automated. These observations led us to support end-user programmability of the Rigi editor through a scripting language. Users can now write scripts to increase the flexibility of the editor. Complex decomposition or layout tasks are automated for more consistency. User interfaces and interactions may be tailored as desired. The editor is extensible, using the scripting language to transparently integrate existing capabilities.

Previously, the user interface and kernel of the editor were tightly coupled. We added a transparent middle layer to allow scripting of all editor operations. Instead of writing yet another command language, we defined the Rigi Command Language (RCL) using Tcl [24]. Tcl provides a powerful, extendible core language and was specifically designed to be embedded into applications. Since the implementation is interpreted, there is no need to recompile an application—very important to ease experimentation during program understanding tasks. Moreover, scripts are easy to write and are often fairly short. Tcl is application-independent and provides two interfaces: a textual one to users who issue Tcl commands, and a procedural one to the host application. Using the procedural interface, Rigi implements and registers new commands that are indistinguishable from built-in Tcl commands. We added commands to access the internal Rigi state, such as the canvas size and graph model, as well as commands to trigger the built-in Rigi operations (also accessible through pull-down and pop-up menus). RCL scripts have been written in terms of these commands to provide additional algorithms for off-line decomposition, analysis, and visualization.

Integration is one particularly important benefit of scripting. Tcl supports inter-application communication between Tcl-based tools as a possible avenue for integration. As well, Tcl provides primitives to access and coordinate external tools, even to those interactive tools that were not written with scripting control in mind. Thus, the toolset available to the analyst is effectively unlimited. For example, visualizing dependencies is a common operation during program understanding. Scripts were written to access external tools for spring [12] and Sugiyama [27] graph layout. The following is an RCL script for applying a spring layout algorithm to the graph in a Rigi window. The registered names of the new commands are `layout` and `spring`.

```
# call a graph layout program on the subgraph
# in the current window and along the given
# arc type
proc layout { program {window 0} {arctype any} } {
    if {$window == 0} {
        # no window given; get current window
        set window [get_window_id]
    }

    # compute names of temporary files
    set graphin [format "/tmp/%s-in" $window]
    set graphout [format "/tmp/%s-out" $window]

    # write subgraph, call layout program,
    # read result
    writeGEF $graphin $window $arctype
    exec $program < $graphin.gef > $graphout.gef
    readGEF $graphout $window

    # delete temporary files
    exec rm $graphin.gef
    exec rm $graphout.gef
}

# run spring layout algorithm on the subgraph
# in the current window and along the given
# arc type
proc spring { {window 0} {arctype any} } {
    if {$window == 0} {
        # no window given; get current window
        set window [get_window_id]
    }

    if {[is_connected $window $arctype]} {
        # subgraph is connected;
        # call spring layout
        layout gel-spring $window $arctype
    } else {
        open_message_panel "Graph not connected"
    }
}
```

The `spring` script uses the `layout` command. The `layout` script uses the `writeGEF` script command to convert the subgraph of the Rigi graph model portrayed in the active window to a file conforming to the GEF format expected by a GraphEd tool called `gel_spring` [15]. The `layout` script then executes `gel_spring` and reads the resulting new layout back into the editor using the `readGEF` command (which also adjusts the positions of nodes in the graph model). The resulting layout from `spring` is based on physical properties of attraction and repulsion, and may help ease the identification of candidate subsystems.

The addition of a scripting language migrates the Rigi editor closer to a highly flexible visualization engine. The editor is a reusable component that can be adapted to new application domains (and, hence, is domain retargetable). This adaptability is important for presenting the varied program understanding analyses expected in the RevEngE project, such as clone detection, data bindings, defect detection, and subsystem structure identification in a visually integrated way. Moreover, to be better accessible to other tools, the format for the tuple streams through which the current Rigi components communicate was standardized and better defined. The editor was also decoupled from the existing workspace to support the input of these tuple streams as flat files. This capability provides another avenue of data interchange with other tools.

Presently, the Rigi parser supports C, C++, COBOL, LaTeX, and, to a certain extent, PL/AS. One goal was to extend the PL/AS part by analyzing data types, data dependencies, structures, call dependencies, and pointers (references). Our original intent was to transform PL/AS source code into something very similar to the C language so that the Rigi C parser would be able to analyse it. Instead, we decided to take advantage of Rigi's ability to read a stream of tuples representing relations and objects. These tuple streams could be generated from compiler listings produced by PL/AS, where information about program structure is presented in a more regular format than the source code itself. AWK [1], a well-known UNIX file-processing tool, seemed suitable, and we wrote AWK scripts to generate the required tuples from a compiler listing. For example:

```
DATA   file-name   data-name
CALL   file-name   procedure-name
PROC   file-name   procedure-name
STRUCT file-name   struct-name
MEMBER struct-name member-name
```

`DATA` This tuple represents the declaration of a named data item in a file.

`CALL` This tuple is created for each call in a module to a procedure that is not defined in the same module.

`PROC` This tuple is created for each call in a module to an externally visible procedure that is defined in the same module. No tuples were created for calls to PL/AS built-in functions, which are usually specific sequences of 370 assembler instructions.

`STRUCT` This tuple is created for each compound data type (array, record, etc.) found in a source file.

`MEMBER` This tuple is created for each member of a structure. The naming conventions used in SQL/DS allow the assumption that different instances of a structure name actually represent the same structure. These occurrences are rare.

These tuples were loaded into Rigi's repository to produce a visual representation of the SQL/DS system. Figure 3 depicts a spring layout of a (logical) SQL/DS subsystem (the routines associated with adding a foreign key to the database, within the physical component `ARIXI`). In this logical subsystem, three modules (`ARIXIGK`, `ARIXIAP`, `ARIXIUK`) are shown as well as some of the structures and data variables they access. A fourth module (`ARIXIAF`) is filtered from the view for clarity. A spring layout easily shows the data being shared by the modules. Nevertheless, deducing such a logical subsystem in SQL/DS at this level of abstraction (neither too high nor too low) is very difficult because of the structural complexity of the software. Thus, there is a need to integrate with and access alternative clustering methods.

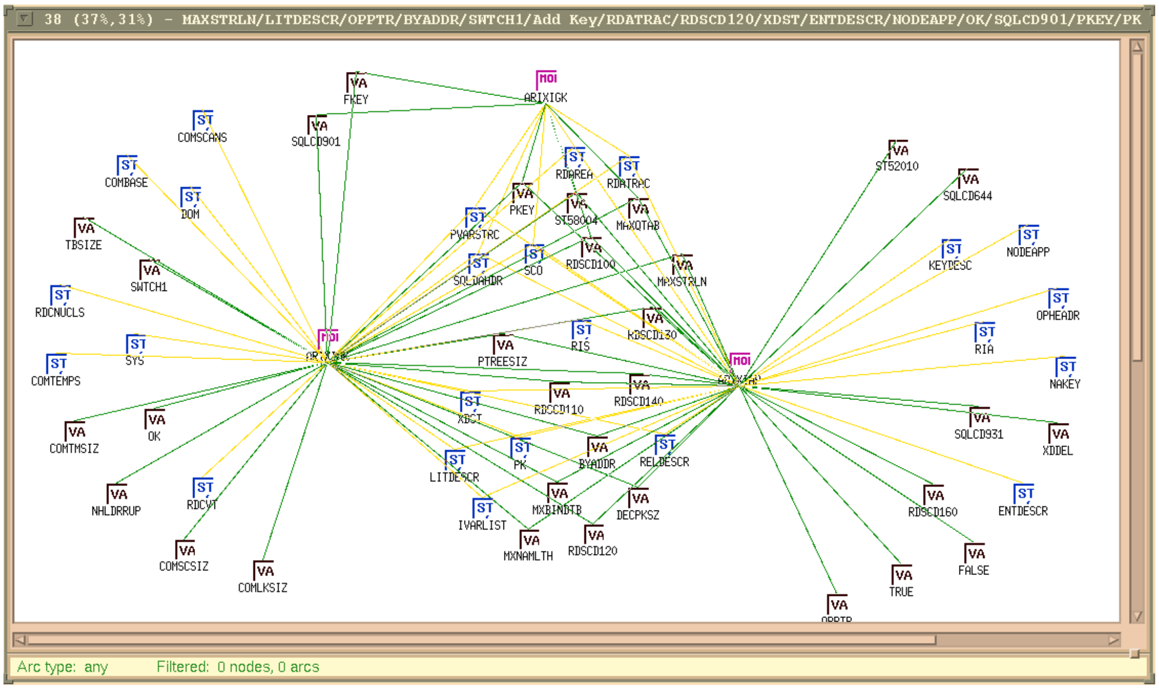

Figure 3: Spring layout of an SQL/DS subsystem

4 Pattern Matching in REFINE

Pattern matching is involved in every major aspect of program understanding. McGill's contribution to the project has focused on integrating novel pattern-matching algorithms for program analysis and design discovery into the environment. Adopted is a flexible, generalized, pattern-matching paradigm that does not limit maintainers to a fixed plan library [16]. Of interest are algorithms to identify useful patterns efficiently and visualize pattern matches effectively.

As software is maintained, patches of code tend to be introduced that form hidden relationships in the program. Subsystem clustering attempts to group non-contiguous program fragments that should be considered together. Two fragments should be clustered together if and only if one fragment has, or is likely to have, a significant effect on the behaviour of the other. To date, we have considered structural elements that indicate the exchange of program resources, such as data bindings and common references [17]. A data binding is a triplet consisting of two program fragments and a variable where the variable is set in one fragment and used in the other. A common reference is where two or more fragments use or update a particular variable.

We used REFINE to implement a prototype clustering algorithm that considers the number of common references among arbitrary sets of fragments and a clustering algorithm based on data bindings on the set of variables in the software. The program representation scheme is an object-oriented annotated abstract syntax tree; a grammar is used for parsing and a domain model is defined to specify the object hierarchies. During analysis, the tree is annotated with information on system structure, data flow, and links to informal information (for example, local variables, global variables, functions call, and aliasing).

The results of the analysis can be displayed in tabular form within REFINE or passed to the repository for display by Rigi. Figure 4 presents the tables of clusters based on data bindings and common references that REFINE generated for

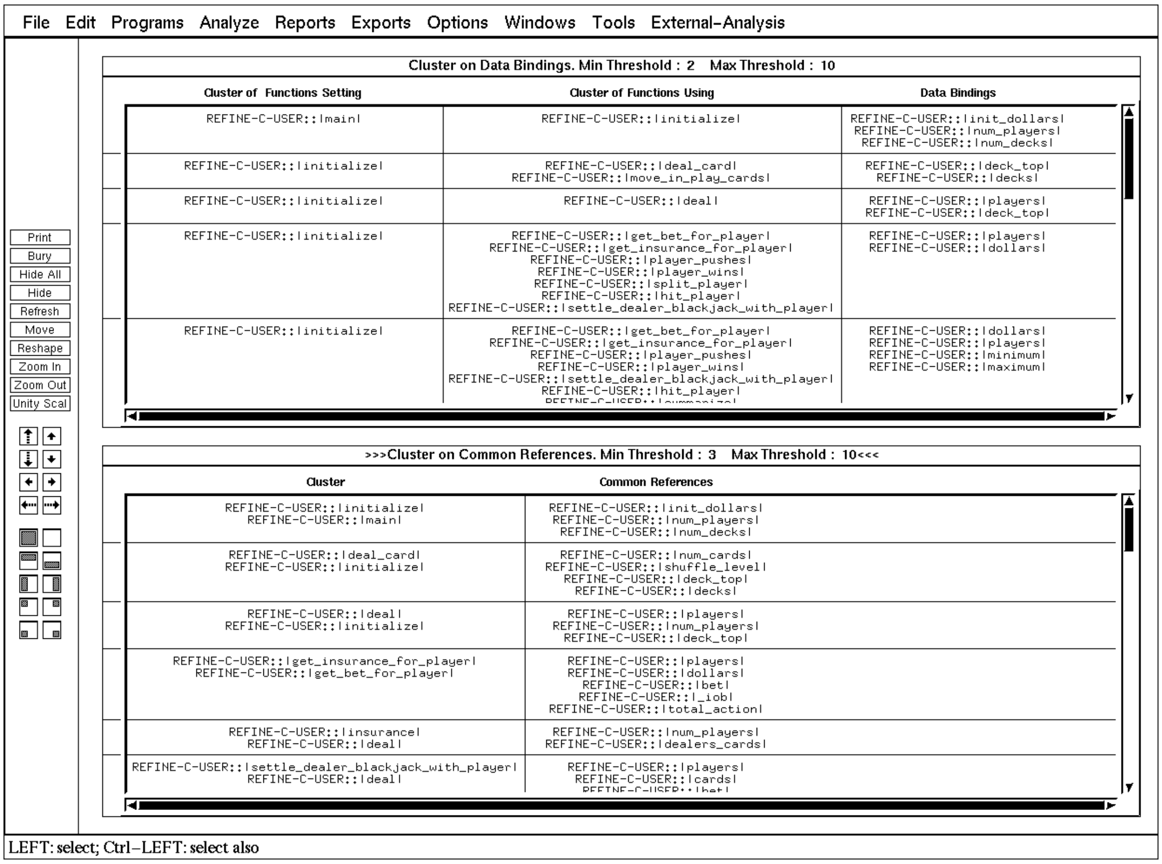

Figure 4: REFINE data bindings and common references tables

a game program written in C.

The availability of the clustering algorithms assist the task of subsystem composition in Rigi. Figure 5 depicts the original resource flow graph that Rigi's parsing subsystem extracted from the same source code. There are icons for the functions and a data type called `player`; the arcs represent function calls or data accesses. REFINE can augment this information by extracting the variables as well, computing clusters, and loading this analysis into the repository. The process of subsystem composition in Rigi can subsequently be guided by these clusters. A view of clustering based on data bindings is shown in Figure 6. The middle column are variables extracted by REFINE and the left and right columns represent clusters of functions that set or use, respectively, those variables.

The static analysis is non-trivial because of the aliasing problem in languages such as C and PL/AS. The current implementation of the clustering algorithms cannot detect relationships due to aliasing. Instead, it examines global variables and parameters of functions. This work is being extended to integrate aliasing information from McGill's Compiler Architecture Testbed (McCAT) [14]. McCAT is a research compiler for C, designed to test techniques for generating optimized code on modern RISC and parallel architectures. One key element of the compiler is its approach for interprocedural analysis using "points-to" information [11].

5 Related Work

SoftBench from Hewlett Packard is a platform that integrates tools via a mechanism introduced in the Field programming environment

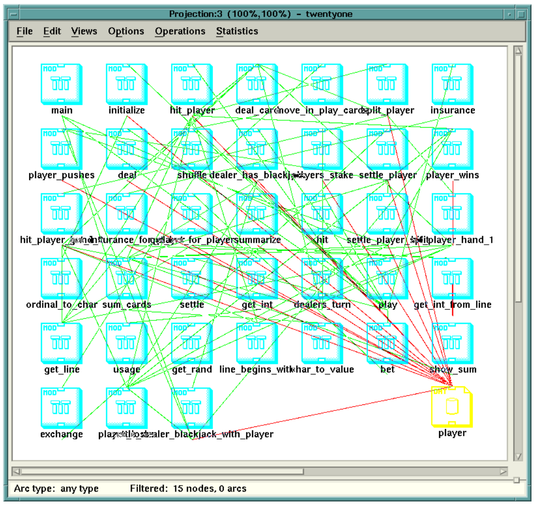

Figure 5: Rigi resource flow graph

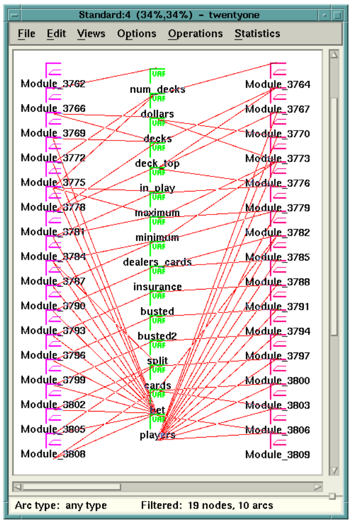

Figure 6: Clustering based on data bindings

[25]. This mechanism allows tools to interact by sending messages to a message server that selectively broadcasts them to those tools that have previously registered a need to be notified. For close integration with other tools, a new tool must provide a programmatic message-based interface in addition to its user interface; it must support request messages for services it provides and send notification messages announcing the actions it has performed.

PCTE Workbench from VISTA Technologies [2] is a toolkit for constructing hypermedia-based environments and applications. It is based on the Portable Common Tool Environment (PCTE), which provides an extensible structure for tool integration and the construction of software development environments [4]. Data integration is based on the PCTE Object Management System [13], which supports a hypertext-like data model. Control integration is provided by broadcast messaging built around an interpreter for an object-oriented, Lisp-based scripting language.

Software through Pictures (StP)[3] [30] from Interactive Development Environments is an integrated software-engineering environment that provides an open architecture, data sharing through a central repository, a common graphical editor, and customization through a scripting language. The editor is tailorable to a particular software engineering methodology through rule files, and provides presentation integration by showing a similar and consistent user interface for all methodologies structured or object-oriented.

Work on software repositories includes [9, 10], which discusses a knowledge representation system based on the language CLASSIC, which is used to describe software objects. CLASSIC is also used by [26] as the representation language for a communications software repository. This repository includes information about the software code, its intended function, and discovered relationships between them. Finally, [8] presents a similar system based on Telos, which stores requirements, design, and implementation information about information systems, and is intended for reuse-oriented application development.

6 Summary

This paper described an implementation for integrating tools developed to support program understanding activities. The strategy is based on a software repository using a rich and extensible global schema to integrate the types of information handled by a given toolset. The architecture supports a software bus intended to

[3]Software through Pictures is a registered trademark of Interactive Development Environments

accommodate communication among tools and the repository. In addition, the paper reports on work with two existing tools, Rigi and REFINE, so that they can be interfaced with the architecture. Finally, it reports on recent work within the project on tools for program analysis and design recovery.

We are only beginning to discover the benefits of integration. For example, Rigi has a much more flexible and robust user interface for visualization than REFINE, but it lacks REFINE's strengths for detailed syntactic analysis on the syntax tree of a program. Trying to implement these capabilities for detailed analysis in the scripting language of Rigi would be redundant and perhaps even inadequate. Cloning analysis is another good example where REFINE can perform the analysis and Rigi can visually highlight the clones using customizable icons, color schemes, and placement. An integration architecture with a global schema is needed to bring tools with such varied strengths and capabilities together to form a unified program understanding toolset—unified in terms of data and control integration.

About the Authors

John Mylopoulos *Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario, Canada M5S 1A4.* `jm@ai.utoronto.ca`. Dr. Mylopoulos is a professor of computer science at the University of Toronto. He received his Ph.D. degree from Princeton University in 1970. His research interests include knowledge representation and conceptual modelling, covering languages, implementation techniques for large knowledge bases, and the application of knowledge bases to software repositories. He is currently leading a number of research projects and is principal investigator of both a national and a provincial Centre of Excellence for Information Technology. His publication list includes more than 120 refereed journal and conference proceedings papers and three edited books. He is the recipient of the first ever Outstanding Services Award given out by the Canadian AI Society (1992), and also a co-recipient of a best paper award at the *16th International Conference on Software Engineering.*

Martin Stanley *Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario, Canada M5S 1A4.* `mts@ai.utoronto.ca`. Mr. Stanley received his M.Sc. degree in computer science from the University of Toronto in 1987. His research interests include knowledge representation and conceptual modeling, with particular application to the building of software repositories. He is currently a research associate in the Department of Computer Science at the University of Toronto, with primary responsibility for the Toronto portion of the RevEngE project.

Kenny Wong *Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, BC, Canada V8W 3P6.* `kenw@csr.uvic.ca`. Mr. Wong is a Ph.D. candidate in the Department of Computer Science at the University of Victoria. He worked in the program understanding project while at CAS during the summer of 1993 and 1994. His research interests include program understanding, runtime analysis, user interfaces, object-oriented programming, and software design. He is a member of the ACM, USENIX, and the Planetary Society.

Morris Bernstein *School of Computer Science, McGill University, 3480 University Street, Room 318, Montréal, Québec, Canada H3A 2A7.* `zaphod@cs.mcgill.ca`. Mr. Bernstein received his B.Sc. and M.Sc degrees from McGill University. His research interests include software development, program understanding, compiler design, and application-domain languages. He is currently a research assistant with primary responsibility for the McGill portion of the RevEngE project.

Renato De Mori *School of Computer Science, McGill University, 3480 University Street, Room 318, Montréal, Québec, Canada H3A 2A7.* `demori@cs.mcgill.ca`. Dr. De Mori received a doctorate degree in Electronic Engineering from Politecnico di Torino, Italy, in 1967. Since 1986, he has been a professor and the director of the School of Computer Science at McGill University. In 1991, he became an associate of the Canadian Institute for Advanced Research and project leader of the Institute for Robotics and Intelligent Systems, a

Canadian Centre of Excellence. His current research interests are stochastic parsing techniques, automatic speech understanding, connectionist models, and reverse engineering. He is the author of many publications in the areas of computer systems, pattern recognition, artificial intelligence, and connectionist models. He is on the board of the following international journals: the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *Signal Processing*, *Speech Communication*, *Pattern Recognition Letters*, *Computer Speech*, and *Language and Computational Intelligence*. He is a fellow of the IEEE Computer Society.

Graham Ewart *Centre for Advanced Studies, IBM Software Solutions Toronto Laboratory, 844 Don Mills, North York, ON, Canada M3C 1V7.* `ewart@vnet.ibm.com`. Mr. Ewart is a senior development analyst of the IBM Software Solutions Toronto Laboratory and is currently the principal investigator for the program understanding project at CAS. Prior to joining CAS, he was the lead architect for the IBM C/370 family of compilers and runtimes. His research interests include software maintenance, program understanding, document understanding, and reverse engineering.

Kostas Kontogiannis *School of Computer Science, McGill University, 3480 University Street, Room 318, Montréal, Québec, Canada H3A 2A7.* `kostas@binkley.cs.mcgill.ca`. Mr. Kontogiannis received a B.Sc degree in Mathematics from the University of Patras, Greece, and an M.Sc degree in Artificial Intelligence from Katholieke Universiteit Leuven in Belgium. Currently, he is a Ph.D candidate in the School of Computer Science at McGill University. His interests include plan localization algorithms, software metrics, artificial intelligence, and expert systems.

Ettore Merlo *Département de Génie Électrique (DGEGI), École Polytechnique de Montréal, C.P. 6079, Succ. Centre Ville, Montréal, Québec, Canada H3C 3A7.* `merlo@rgl.polymtl.ca`. Dr. Merlo graduated from the University of Turin, Italy, in 1983 and obtained a Ph.D. degree in computer science from McGill University in 1989. From 1989 until 1993, he was the lead researcher of the software engineering group at the Computer Research Institute of Montreal. He is currently an assistant professor of computer engineering at École Polytechnique de Montréal, where his research interests include software re-engineering, software analysis, and artificial intelligence. He is a member of IEEE Computer Society.

Hausi A. Müller *Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, BC, Canada V8W 3P6.* `hausi@csr.uvic.ca`. Dr. Müller is an associate professor in the Department of Computer Science at the University of Victoria, where he has been since 1986. He received his Ph.D. in computer science from Rice University in 1986. From mid 1992 to mid 1993, he was on sabbatical at CAS, working in the program understanding project. His research interests include software engineering, software analysis, reverse engineering, re-engineering, programming-in-the-large, software metrics, and computational geometry. He is currently a Program Co-Chair of the *International Conference on Software Maintenance* (ICSM '94) in Victoria and the *International Workshop on Computer Aided Software Engineering* (CASE '95) in Toronto. He is a member of the editorial board of *IEEE Transactions on Software Engineering*.

Scott R. Tilley *Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, BC, Canada V8W 3P6.* `stilley@csr.uvic.ca`. Mr. Tilley is currently on leave from the IBM Software Solutions Toronto Laboratory, and is a Ph.D. candidate in the Department of Computer Science at the University of Victoria. His first book on home computing was published in 1993. His research interests include end-user programming, hypertext, program understanding, reverse engineering, and user interfaces. He is a member of the ACM and the IEEE Computer Society.

Marijana Tomic *Centre for Advanced Studies, IBM Software Solutions Toronto Laboratory, 844 Don Mills, North York, ON, Canada M3C 1V7.* `mtomic@vnet.ibm.com`. Mrs. Tomic is a post-doctoral student from the University of Victoria, working in the program understanding project at CAS. Her research interests include software engineering in general, and software maintenance, program understanding, reverse engineering, re-engineering, and re-structuring in particular.

References

[1] Aho, A., Kernighan, B., and Weinberger, P. *The AWK Programming Language.* Addison-Wesley, 1988.

[2] Arora, A.K., Hurst, D.W., and Ferrans, J.C. "Building Diverse Environments with PCTE Workbench", Proc. PCTE '93, 1993.

[3] Arnold, R.S. "Tutorial on Software Re-Engineering", IEEE International Conference on Software Maintenance, (ICSM '90), San Diego, November 1990.

[4] Boudier, G., Gallo, F., Minot, R., and Thomas, I. "An Overview of PCTE and PCTE+", *ACM SIGSOFT Software Engineering Notes*, 13(5), February 1989.

[5] Buss, E. and Henshaw, J. "Experiences in Program Understanding", Proc. CASCON '92, Toronto, November 1992.

[6] Buss, E. et al. "Investigating Reverse Engineering Technologies: The CAS Program Understanding Project", *IBM Systems Journal*, 33(3), 1994.

[7] Carroll, A. *ConversationBuilder: A Collaborative Erector Set.* Ph.D. Thesis, University of Illinois, 1993.

[8] Constantopoulos, P., Jarke, M., Mylopoulos, J., and Vassiliou, Y. "The Software Information Base: A Server for Reuse", *The VLDB Journal* (to appear).

[9] Devanbu, P., Selfridge, P., Ballard, B., and Brachman, R.P. "Steps Towards a Knowledge-Based Software Information System", Proc. International Joint Conference on Artificial Intelligence, Detroit, August 1989.

[10] Devanbu, P., Brachman, R.P., Selfridge, P., and Ballard, B. "A Classification-Based Software Information System", Proc. IEEE International Conference on Software Engineering, (ICSE 12), May 1990.

[11] Emami, M., Ghiya, R., and Hendren, L.J. "Context-Sensitive Interprocedural Points-to Analysis in the Presence of Function Pointers", Proc. Conference on Programming Language Design and Implementation, (SIGPLAN '94), Orlando, June 1994.

[12] Fruchtermann, T. and Reingold, E. "Graph Drawing by Force-Directed Placement", Technical Report UIUC CDS-R-90-1609, Department of Computer Science, University of Illinois at Urbana-Champaign, 1990.

[13] Gallo, F., Minot, R., and Thomas, M.I. "The Object Management System of PCTE as a Software Engineering Database Management System", Proc. Second ACM Symposium on Practical Software Development Environments (SIGSOFT '86), Palo Alta, December 9–11, 1986.

[14] Hendren, L.J., Donawa, C., Emami, M., Gao, G.R., Justiani, and Sridharan, B. "Designing the McCAT Compiler Based on a Family of Structured Intermediate Representations", ACAPS Technical Memo 46, School of Computer Science, McGill University, Montreal, 1992.

[15] Himsolt, M. "GraphEd: The Design and Implementation of a Graph Editor", GraphEd Distribution Kit, 1993.

[16] Kontogiannis, K.A., De Mori, R., Bernstein, M. and Merlo, E. "Localization of Design Concepts in Legacy Systems", Proc. IEEE International Conference on Software Maintenance, (ICSM '94), Victoria, BC, September 19–23, 1994 (to appear).

[17] Kontogiannis, K.A., Tilley, S.R., De Mori, R. and Müller, H.A. "User-Assisted Design Recovery for Legacy Software Systems", Workshop on Software Engineering and Artificial Intelligence in IEEE International Conference on Software Engineering, (ICSE 16), Sorrento, Italy, May 16–17, 1994.

[18] Müller, H.A. *Rigi—A Model for Software System Construction, Integration, and Evolution Based on Module Interface Specifications*, Ph.D. Thesis, Rice University, August 1986.

[19] Müller, H.A. "(k,2)-Partite Graphs as a Structural Basis for the Construction of Hypermedia Applications", Technical Report DCS-119-IR, University of Victoria, June 1989.

[20] Müller, H.A. and Corrie, B.D. "Measuring the Quality of Subsystem Structures". Technical Report DCS-193-IR, University of Victoria, November 1991.

[21] Müller, H.A., Tilley, S.R., Orgun, M.A., Corrie, B.D., and Madhavji, N.H. "A Reverse

Engineering Environment Based on Spatial and Visual Software Interconnection Models", *ACM SIGSOFT Software Engineering Notes*, 17(5), December 1992.

[22] Müller, H.A., Orgun, M.A., Tilley, S.R., and Uhl, J.S. "A Reverse Engineering Approach to Subsystem Structure Identification", *Journal of Software Maintenance: Research and Practice*, 5(4), December 1993.

[23] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. "Telos: Representing Knowledge About Information Systems", *ACM Transactions on Information Systems*, 8(4), October 1990.

[24] Ousterhout, J.K. *An Introduction to Tcl and Tk*. Addison-Wesley, 1994.

[25] Reiss, S.P. "Interacting with FIELD", *Software—Practice and Experience*, 20, June 1990.

[26] Selfridge, P. "Knowledge Representation Support for a Software Information System", Proc. Seventh IEEE Conference on AI Applications, February 1991.

[27] Sugiyama, K., Tagawa, S., and Toda, M. "Methods for Visual Understanding of Hierarchical Systems", *IEEE Transactions on Systems, Man, and Cybernetics*, 11(4), 1981.

[28] Tilley, S.R., Müller, H.A., and Orgun, M.A. "Documenting Software Systems with Views", Proc. 10th International Conference on Systems Documentation, (SIGDOC '92), Ottawa, October 13–16, 1992).

[29] Tilley, S.R., Müller, H.A., Whitney, M.J., and Wong, K. "Domain-Retargetable Reverse Engineering", Proc. 1993 International Conference on Software Maintenance, (ICSM '93), Montreal, September 27–30, 1993.

[30] Wasserman, A.I. and Pircher, P.A. "A Graphical, Extensible Integrated Environment for Software Development", Proc. Second ACM Symposium on Practical Software Development Environments (SIGSOFT '86), Palo Alta, December 9–11, 1986.