POSITION PAPER DOMAIN ANALYSIS AND REVERSE ENGINEERING

Spencer Rugaber

Georgia Institute of Technology
spencer@cc.gatech.edu

1. THE PROBLEM

Reverse engineering takes a program and constructs a high level representation useful for documentation, maintenance, or reuse. To accomplish this, most current reverse engineering techniques begin by analyzing a program's structure. The structure is determined by lexical, syntactic, and semantic rules for legal program constructs. Because we know how to do these kinds of analyses quite well, it is natural to try and apply them to understanding a program.

But knowledge of program structures alone is insufficient to achieve understanding, just as knowing the rules of grammar for English are not sufficient to understand essays or articles or stories. Imagine trying to understand a program in which all identifiers have been systematically replaced by random names and in which all indentation and comments have been removed [2]. The task would be difficult if not impossible.

The problem is that programs have a purpose; their job is to compute something. And for the computation to be of value, the program must model or approximate some aspect of the real world. To the extent that the model is accurate, the program will succeed in accomplishing its purpose. To the extent that the model is comprehended by the reverse engineer, the process of understanding the program will be eased. In order to understand a program, therefore, it makes sense to try and understand its context: that part of the world it is modeling.

Given that the source code by itself is not sufficient to understand the program and given that traditional forms of documentation are not likely to provide the information needed, the question arises whether there is an alternate approach better suited to the needs of reverse engineering. This essay argues that application domain modeling provides such an approach.

2. DOMAIN ANALYSIS

Domains

A *domain* is a problem area. Typically, many application programs exist to solve the problems in a single domain. Arango and Priéto-Diaz [1] give the following prerequisites for the presence of a domain: the existence of comprehensive relationships among objects in the domain, a community interested in solutions to the problems in the domain, a recognition that software solutions are appropriate to the problems in the domain, and a store of knowledge or collected wisdom to address the problems in the domain.

Once recognized, a domain can be characterized by its vocabulary, common assumptions, architectural approach, and literature.

- The problems in a domain share a common vocabulary. In the income tax domain, terms like "adjusted gross income," "dependent," and "personal exemption" are commonly used.
- The programs that solve problems in a domain may also share common assumptions or tactics. For example, in the income tax domain it is understood that there are multiple places where the same information, such as adjusted gross income, must be supplied and that all of these sites must be altered when any one of them is changed.
- It may be the case that a common architectural approach is used to solve problems in a domain. In the income tax example, a given computation will likely obtain its operands from the results of other computations. The set of computations and the dependencies among them form a partial order, and programs in this domain are likely to be structured in a way to maintain and take advantage of this ordering.

- A domain exists independently of any programs to solve its problems. It likely has its own literature and experts. For example, there are many "how-to" books in the income tax domain, and experts seem to pop up like weeds every spring.

Domain Analysis

According to Neighbors [7], *domain analysis* "is an attempt to identify the objects, operators, and relationships between what domain experts perceive to be important about the domain." As such, it bears a close resemblance to traditional systems analysis, but at the level of a collection of problems rather than a single one.

Domain engineering/modeling/analysis is an emerging research area in software engineering. It is primarily concerned with understanding domains in order to support initial software development and reuse, but its artifacts and approaches will prove useful in support of reverse engineering as well.

Domain Representation

In order for domain analysis to be useful for software development, reuse, or reverse engineering, the results of the analysis must be captured and expressed, preferably, in a systematic fashion. Among the aspects that might be included in such a representation are domain objects and their definitions, including both real world objects like "tax rate tables" and concepts like "long term capital gains"; solution strategies/plans/architectures like "partial order of computation"; and a description of the boundary and other limits to the domain like "federal, personal income tax return." An unresolved issue, of importance both to software developers and reverse engineers, is the exact form of the representation and the extent of its formality.

Relationship to Reverse Engineering

What role might a domain description play in reverse engineering a program? In general, a domain description can give the reverse engineer a set of expected constructs to look for in the code. These might be computer representations of real world objects like tax rate tables or deductions. Or they may be algorithms, such as the LIFO method of appraising inventories. Or they might be overall architectural schemes, such as a data flow architecture for implementing the computational partial order described above.

Because a domain is broader than any single problem in it, there may be expectations engendered by the domain representation that are not found in a specific program (the inventory algorithm may not appear in a program to compute personal income taxes but might in a business tax program). Because a program is not always accurate or up-to-date, there may be things missing or incorrectly expressed in the program, despite contraindications in the domain representation. And, because a program is often used for more than one purpose, it may include components that do not appear at all in the domain representation, such as a checkbook balancing feature in an income tax package.

Nevertheless, a domain representation can establish expectations to be confirmed in a program. Furthermore, the objects in the domain representation are related to each other and organized in prototypical ways that may likewise be recognized in the program. Hence, a domain representation can act as a schema for controlling the reverse engineering process and a template for organizing its results.

3. ISSUES

Many questions arise concerning how best to make use of domain analysis in support of reverse engineering. The questions can be partitioned into the areas of methodology, representation and tools.

Methodology

1. Perhaps the overriding question of this research is whether domain analysis can help in the reverse engineering process at all. Clearly, this essay assumes so, but the assumption needs to be validated. The projects described in the next section are intended to explore this question.
2. Corollary to this is the question of how best to make use of the domain knowledge obtained. For example, even if we imagine existing, complete, well-organized descriptions for each of the domains related to the income tax program, it is not clear how best to use them to understand a program. Which one should we start with? How do we coordinate a search for multiple expected constructs derived from several domains?

3. A subsidiary methodological issue concerns knowledge of the domain learned while examining a program. We would like domain descriptions to grow and become more complete over time, but domain descriptions need to be definitive, and the reverse engineer need not be a knowledge engineer nor have sufficient expertise to judge the accuracy, relevance, and placement of the new information in the domain description.

Representation

1. The fundamental question concerning representation is what is the best form for a domain description to take in order to support reverse engineering, or whether, in fact, a single, "best" representation can be devised [3]. Certainly, domain theorists do not yet agree on how to represent domain information, but a consistent representation is a prerequisite to broadly applicable tools.
2. Related to this question is the issue of how much formality a domain representation should entail. Many of the domain models in the literature use sophisticated mathematical techniques. Not only does this present a barrier to some potential users, but it raises the question of how best to deal with informal information, such as the heuristic that indicates not to investigate deducting medical expenses until they form a significant fraction of income. Of course, some degree of formality is a prerequisite for tool support.
3. Another issue concerns the relation of the domain representation to the program description that emerges as a result of the reverse engineering process. If a domain has a natural structure or if programs solving domain problems tend to have a favored architecture, then the program description should somehow mirror this. But what if the program includes several domains, each with their own preferred structures?
4. Several technical questions also exist concerning domain representations. How much detail should they include? How should they deal with optional information? How should they express abstractions such as might arise with a parameterized domain?

Tools

1. Domains are complex. They not only include a lot of information, but the information is highly interrelated. The question then arises of how best to access this information? Are program browser-like tools sufficient? CASE tools? Or is a new approach required?
2. Tools that access domain information may have to do a lot of specialized inferencing, for example, to confirm that a given program contains a valid implementation of some domain concept. What are the implications of this? A variety of inferencing tools exist that can be categorized as trading off power for efficiency. Where on this curve is the right place for domain based reverse engineering tools?
3. An intriguing question pertains to tool generation. Mature domains enable application generation technology, such as report writers. How about the inverse? Can we build application analyzer generators?
4. Finally, what should be done with all the existing reverse engineering tools that do not take advantage of domain knowledge? Can they be adapted or integrated? Need they be?

4. DOMAIN ANALYSIS AND REVERSE ENGINEERING PROJECTS AT GEORGIA TECH

Researchers in the College of Computing at the Georgia Institute of Technology are actively investigating the relationship of domain analysis and reverse engineering through a variety of projects. The projects involve domains ranging from the report writing and inventory domains, to user interface toolkit components, to the kinematics of natural and artificial objects in our solar system.

The TRANSOPEN Project

The Army Research Laboratory has for several years sponsored research at Georgia Tech to investigate the transition of existing Army management information systems from their traditional batch, mainframe environment to an interactive, distributed, workstation, open systems environment. The work includes researchers concerned with communications protocols, database integration, and business process re-engineering. Our part in this project has been concerned with the transition of the actual software, including issues of strategy selection, reverse engineering process, and tool support [4, 8].

Our current work with the project directly involves domain analysis. We have taken a specific domain, report writing, and performed a domain analysis on it. We are experimenting with ways to represent the results of the

analysis with a knowledge representation language so that we can use the representation to drive the reverse engineering of several applications in the domain. We hope to learn several things from this work directly addressing the issues raised in Section 5: To what extent has the domain knowledge aided (or hindered) the reverse engineering process? How should the domain model be most effectively represented? And what tools would facilitate the process? In the future, we hope to look at less well-defined domains and at the issues arising from the use of more than one domain in the same application.

The Knowledge Worker System (KWS) Project

The Army Corps of Engineers Research Laboratory (CERL) has sponsored the development of a personal information system called Knowledge Worker. The original version was developed in the C language for IBM-compatible personal computers running MS-Windows and interfacing to the Oracle Relational Database Management System on a remote server. Now CERL is interested in an Ada version for a POSIX workstation running Motif. We have been advising them on the transition process [5]. Of particular interest to us is the issue of re-engineering the MS-Windows user interface to work with Motif [6]. On the surface it might appear that one has merely to textually replace MS-Windows library routine names with Motif ones. Unfortunately, however, things are not so simple. Not only is there not a direct match between the libraries, but there are subtle architectural differences between the two toolkits.

For example, in MS-Windows an option exists whether the application program or the user interface run-time library is responsible for visually indicating that a button has been pressed. In Motif, only the latter option is available. Conversely, in MS-Windows a button exists that can be in one of three different states; in Motif, all buttons are limited to at most two states. Finally, and most troublesome, Motif widgets are arranged hierarchically—specialized widgets inherit features and functions from more general widgets. In MS-Windows, each widget must supply all of its own features and functions. Difficulties like these significantly complicate the problem of selecting appropriate surrogates when adapting an application to a new windowing system.

Commercial vendors have tried to solve these problems, but the users we spoke with were dissatisfied with their products, complaining that either they handled only the superficial translation aspects or they required the engineer to describe the interface in a proprietary language—a non-trivial effort that approximates the effort required to do the translation directly.

Our approach involves a deeper understanding of user interface toolkits and widgets. In fact, we found ourselves modeling such devices as part of a domain. We began by using CLASSIC to describe a part of the Motif widget set and then presented it with a description of an MS-Windows widget taken from KWS. CLASSIC was able to automatically suggest appropriate Motif widgets to use. We have since grown the model to include generic end-user interface requirements resulting in a comprehensive toolkit-independent representation of the domain. Because CLASSIC allows arbitrary LISP procedures to be invoked when an inference is made, automatic code translation is enabled.

Our current work on this project involves dealing with the architectural issues mentioned above and extending our model to deal with other classes of widgets. Also, we intend to try this approach on applications with no graphical interface at all, such as those using character-oriented window libraries, like CURSES, or textual, command language interfaces.

NAIF Library Project

Our newest project is sponsored by the NASA Ames Research Laboratory. NASA researchers are supporting efforts at the Jet Propulsion Laboratory (JPL) to build scientific/engineering applications involved with satellites and other space missions. An example application concerns the sending of messages from a ground station on Earth, via a relay satellite, to a space vehicle in orbit around Mars.

Currently, an extensive library, called the Navigation Ancillary Information Facility (NAIF) SPICELIB library, exists that can be used by scientists at JPL to help construct such applications. The library is written in Fortran and consists of 600 routines dealing with issues such as frame of reference translation, speed of light delays, and ephemeris data. However, the library is not as useful as it should be, and JPL would like to improve retrieval and composition of appropriate subroutines. In particular, they would like to have a formal library specification, and to do so requires reverse engineering the library.

What makes the project particularly interesting to us is that the NASA researchers have constructed a formal, comprehensive, and validated domain model for this class of applications [9]. It is our intent on this project to use the domain model in support of the reverse engineering process; in this case, to incorporate the extensive informal

information provided in the in-line program comments into the domain model.

Other Related Projects at Georgia Tech

We have recently submitted a proposal to an industrial sponsor to support the development of a domain-based program browser (or *dowser*). The point of the work is that providing the reverse engineer with a view of the application domain via a navigational aid will prove more helpful for some maintenance tasks than would a traditional program browser.

We are also looking at the use of formal program semantics as a representational vehicle for reverse engineering information. Although this work is not directly domain related, as mentioned above, reverse engineering representation issues are tightly coupled with those of domain representation.

5. CONCLUSION

The argument for the use of domain analysis in software development is compelling: we need to improve productivity, and to do this, we should reuse as much existing software and its associated documentation as possible. We obtain maximum leverage in reuse by using the highest possible level of abstraction—domain knowledge.

The argument for relating domain analysis to reverse engineering is equally convincing: reverse engineering involves understanding a program and expressing that understanding via a high level representation; understanding concerns both what a program does (the problem it solves) and how it does it (the programming language constructs that express the solution); and the more knowledge we have about the problem, the easier it will be to interpret manifestations of problem concepts in the source code. Based on this logic, I fully expect that any major breakthrough in the automated program understanding and reverse engineering area to take significant advantage of domain information.

References

1. Guillermo Arango and Ruben Priéto-Diaz, ‘‘Domain Analysis Concepts and Research Directions,’’ in *Domain Analysis and Software Systems Modeling*, ed. Ruben Priéto-Diaz and Guillermo Arango, IEEE Computer Society Press, 1991.
2. Ted J. Biggerstaff, ‘‘Design Recovery for Maintenance and Reuse,’’ *IEEE Computer*, vol. 22, no. 7, July 1989.
3. Richard Clayton and Spencer Rugaber, ‘‘The Representation Problem in Reverse Engineering,’’ *Proceedings of the First Working Conference on Reverse Engineering*, Baltimore, Maryland, May 21-23, 1993.
4. Melody Eidbo, Mostafa Ammar, Russ Clark, Rich Clayton, Srinivas Doddapaneni, Rob Dodge, Mike McCracken, Binh Nguyen, Webb Roberts, Steve Rogers, and Spencer Rugaber, ‘‘Transitioning to the Open Systems Environment (TRANSOPEN) Final Report,’’ CIMR - 93-01, Center for Information Management Research, College of Computing, Georgia Institute of Technology, April 14, 1993.
5. Melody Moore, Spencer Rugaber, and Hernan Astudillo, ‘‘Knowledge Worker Platform Analysis Final Report,’’ CIMR - 93-02, Center for Information Management Research, College of Computing, Georgia Institute of Technology.
6. M. Moore and S. Rugaber, ‘‘Issues in User Interface Migration,’’ *Proceedings of the Third Software Engineering Research Forum*, Orlando Florida, November 1993.
7. James M. Neighbors, ‘‘Software Construction from Components,’’ PhD thesis, TR-160, ICS Department, University of California at Irvine, 1980.
8. Spencer Rugaber and Srinivas Doddapaneni, ‘‘The Transition of Application Programs From COBOL to a Fourth Generation Language,’’ *Conference on Software Maintenance - 93*, Montreal, Canada, September 27-30, 1993.
9. Mark Stickel, Richard Waldinger, Michael Lowry, Thomas Pressburger, and Ian Underwood, *Deductive Composition of Astronomical Software from Subroutine Libraries*. Submitted for publication.