Wolf: A tool to recover dataflow oriented designs of software systems

Arun Lakhotia

The Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504
(318) 482-6766, -5791 (Fax)
arun@cacs.usl.edu

Presented in: Fifth Systems Reengineering Technology Workshop, Monterey, CA, February 1995.

Abstract

Dataflow diagrams have been used to model systems, not just software systems, even before computers were invented. Several forward engineering techniques for developing software systems use dataflow information for analyzing the requirements and/or the designs. Wolf is a tool to recover dataflow designs of software systems from their source code. It is expected to benefit problem domains in which dataflow oriented techniques are most suitable (or commonly used) for analyzing a software's requirements and/or design during the forward engineering process. For such domains Wolf will enable the migration of legacy software to CASE tool(s).

1 Introduction

Dataflow diagrams have been used to model systems, not just software systems, even before computers were invented. Several forward engineering techniques for developing software systems use dataflow information for analyzing the requirements and/or the designs. Some such techniques are [2]: De Marco's Structured Analysis and System Specification (SASS), Gane and Sarson's Structured Systems Analysis, Orr's Structured Requirements Definition (SRD), Teichrow's Program Statement Language/Program Statement Analyzer (PSL/PSA), Ross's Structured Analysis and Design Technique (SADT), Ward and Mellor's extension of SASS, and Yourdon's Modern Structured Analysis.

Wolf is a tool to recover dataflow designs of software systems from their source code. It is developed using Reasoning System's Software Refinery and Mark V's ObjectMaker CASE-Tool Products and can recover designs for C programs. The dataflow designs are extracted using Software Refinery and displayed using ObjectMaker. Wolf can display dataflow designs using a variety of notations, including IDEF0 and bubbles. This is made possible by ObjectMaker since it supports most of the dataflow oriented CASE techniques. Though developed for C, Wolf may easily be customized for other languages. This ability is made possible due to using Software Refinery's generic control flow graph abstractions for performing dataflow analysis.

2 Design of Wolf

A *dataflow design* consists of a "hierarchy" of DFDs (or IDEFs) ordered by increasing information flow and processing detail. The top most diagram, the DFD 0, in this hierarchy consists of one bubble representing the entire system and "arrows" showing flow of information from this bubble to "external elements." Each DFD in the hierarchy "decomposes" a bubble of a parent DFD into bubbles denoting its subfunctions. The bubbles that have no decomposition are referred to as "terminal" or "leaf" bubbles.

Internally, Wolf decomposes the problem of recovering dataflow oriented design into the problem of (a) creating layers of processors ("bubbles" or boxes) from a program's functions, (b) identifying the "logical" flow of data between functions, and (c) annotating the flow edges by the information that flows across it. Figure 1 gives a schematic diagram of the approach used by Wolf. Currently Wolf recovers DFDs having only "bubbles" (transformers) and "arrows" (flow of data). The mechanisms to implement "data stores" and access to "external elements" (part of De Marco's DFDs) differ significantly between procedural languages. For instance, in COBOL the physical names of files processed is available in the header of a program; which is not the case in C. Hence we have currently not included information about data stores and external elements.

Ideally, one would like the layer of processors in a recovered design to be as close as possible to that in an expected (or actual) dataflow oriented design. Since the problem of finding the "actual" layer is intractable, Wolf provides multiple methods to create the layers of processors. The simplest method uses the layers induced by a program's call graph. Other methods use numerical

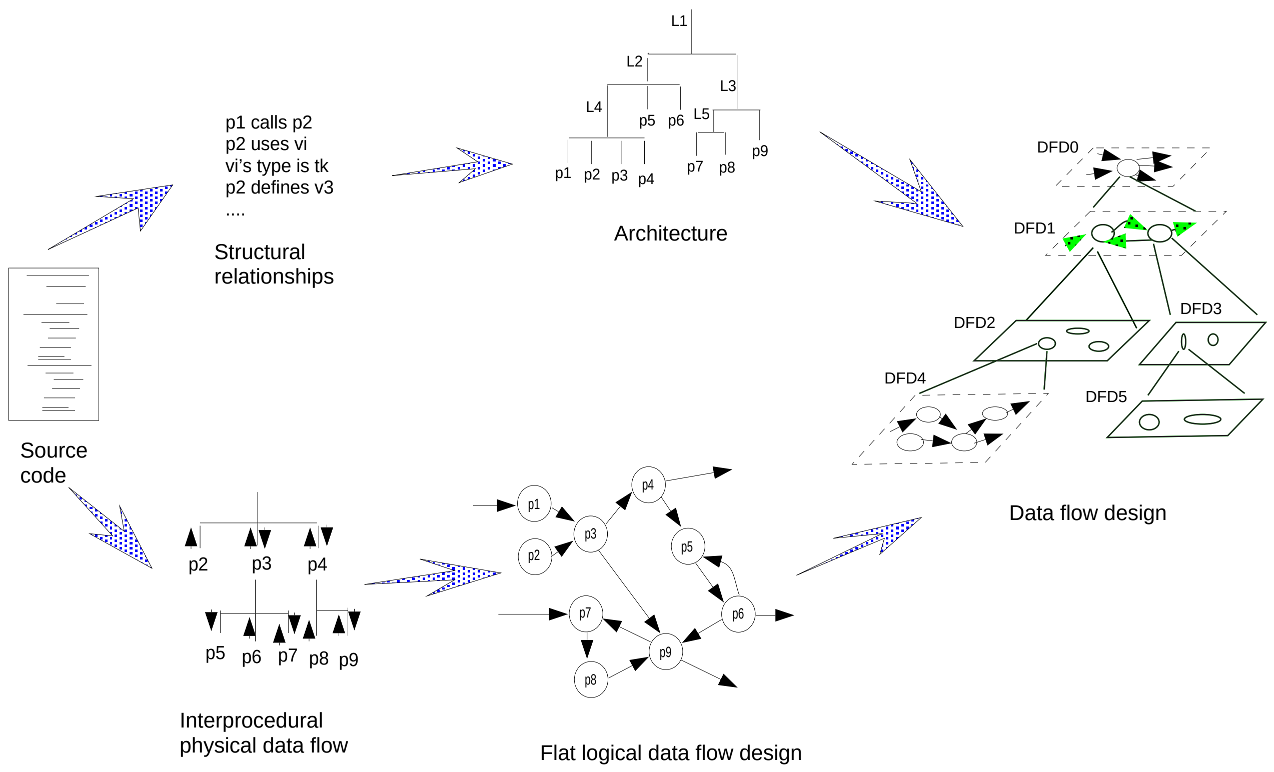

Figure 1 Schematic diagram of the steps used by Wolf for recovering the dataflow design of a software system from its source code.

cluster analysis to group most closely related functions. Wolf thus recovers a variety of dataflow designs for a system. Choosing the "right" design is left to the discretion of the user.

There is a *logical flow of data* from procedure A to procedure B if there is a potential execution path through which a data generated in A is transferred through a sequence of identity assignments to procedure B, and the data is used in B. In other words, if a procedure only acts as a conduit to transfer data between two procedures, without transforming the data, it only participates in the *physical* dataflow. There is a logical flow of data from a procedure that generates data to a procedure that uses it. This definition of *logical* and *physical* dataflow is similar to that used by De Marco in Structured Analysis and System Specification [3]. In a dataflow design we feel one would like to extract the logical flow of data and not the physical flow. Wolf determines such flow information by performing static flow analysis [1].

To complete the recovered dataflow design one also needs to identify the actual information that flows between processing elements. This may be either the name of the variable in the generating procedure whose value is used in the receiving procedure or the "type" of this variable. The various design and analysis methods using DFDs do not specify whether the annotation on the arrows is names of variables or types. Besides, there are trade-offs in either case. Wolf can annotate edges with both the information, the name of a variable or its type. The choice is left to the user.

3 Conclusions

Wolf is the first reverse engineering tool to extract dataflow oriented designs. It is based on the principle that: *For activities (such as reuse and reengineering) requiring an understanding of what a software system does, abstractions that were (or should have been) used in the forward engineering of the system should also be most effective to recover by reverse engineering* [4].

Wolf is expected to be of benefit for problem domains in which dataflow oriented techniques, such as those listed earlier, are most suitable (or commonly used) for analyzing a software's requirements and/or design during the forward engineering process. Some of the activities it may be used for are: maintaining the consistency of design document with its code, migrating/reengineering old software code to emerging Computer Aided Software Engineering (CASE) technology, and for understanding large software systems. Furthermore, the tool may also be used for organizations introducing CASE tools in their development methods. Wolf provides them an easy way to migrate their existing software to the CASE tool(s). Wolf is therefore an important technology in enabling such transition.

Bibliography

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] A. M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice Hall, 1993.

[3] T. De Marco. *Structured Analysis and System Specification*. Yourdon Press, New York, 1978.

[4] A. Lakhotia. What is the appropriate abstraction for understanding and reengineering a software system? *IEEE Computer Society Reverse Engineering Newsletter*, pages 1–2, Sept. 1994.