Architecture Reconstruction to Support a Product Line Effort: Case Study

Liam O'Brien

July 2001

Architecture Tradeoff Analysis

Unlimited distribution subject to the copyright.

Technical Note
CMU/SEI-2001-TN-015

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2001 by Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

Abstract vii

1 Introduction 1

1.1 Architecture Reconstruction Process 1

1.2 The Dali Workbench 1

2 View Extraction 3

3 Database Construction 5

4 View Fusion 6

4.1 Updating the Components Table 6

4.2 Decomposition into High-Level Components 7

5 Architecture Reconstruction 8

6 Results 13

References 14

List of Figures

Figure 1: The Dali Workbench 2

Figure 2: Conversion of the Extracted View to SQL Format 5

Figure 3: Example Fusion Query to Set the Type of Each Element 7

Figure 4: High-Level Component Decomposition for One Motor System 7

Figure 5: Initial Visualization Generated by Loading the View into Rigi 8

Figure 6: File Aggregation of Functions and Variables 9

Figure 7: Components in the INTERNALSW 10

Figure 8: Use of the Blackboard Architectural Style 10

Figure 9: Functional Decomposition of the Components (with COMMIF and EXTERNALSW) 11

Figure 10: Functional Decomposition of INTERNALSW 12

List of Tables

Table 1: Source Statistics for One of the Systems 3

Table 2: The Set of Source Elements and Relations Extracted 4

Abstract

Recently, technical staff members of the SEI performed architecture reconstructions on three small automotive motor systems.[1] One system has an interface to an external bus within the automobile (the Controller Area Network [CAN] bus). The other systems do not have this interface. This technical note describes the architecture reconstruction process that was followed. It provides an overview of the Dali workbench used to support this process, presents the various activities in each phase of the process, and outlines the results that were produced.

[1] The name of the organization sponsoring the case study is not used. Some of the component names also have been changed.

1 Introduction

Members of the technical staff at the SEI recently conducted architecture reconstructions on three small automotive motor systems. One of the systems has an interface to an external bus within the automobile (the CAN bus); the other systems do not include this interface.

An outline of the Architecture Reconstruction process is given below. It is followed by an outline of the Dali workbench, the tool used to support the architecture reconstruction effort.

1.1 Architecture Reconstruction Process

The software architecture reconstruction process comprises five phases, as shown in Figure 1.

1. View Extraction. Extract information from various sources.
2. Database Construction. Convert the extracted information into the Rigi Standard Form a tuple-based data format in the form of “relation <entity1> <entity2>”) [Müller 93]. This format is used to construct an SQL database.
3. View Fusion. Combine views of the information stored in the database.
4. Architecture Reconstruction. Generate an architectural representation by building abstractions and representations of the data.
5. Architecture Analysis. Analyze the resulting architecture. We did not carry out an architecture analysis in this particular case study. More information on Architecture Analysis can be found elsewhere [Kazman 00].

All five phases are highly iterative. In addition, the entire process may have to be repeated several times to extract the right information and to build useful architectural representations.

1.2 The Dali Workbench

We carried out the architecture reconstruction activities using the Dali workbench. It is a collection of tools that primarily include Rigi[1], Dali (an extension of Rigi), and PostgreSQL.[2]

[1] http://www.rigi.csc.uvic.ca/
[2] http://www.postgresql.org

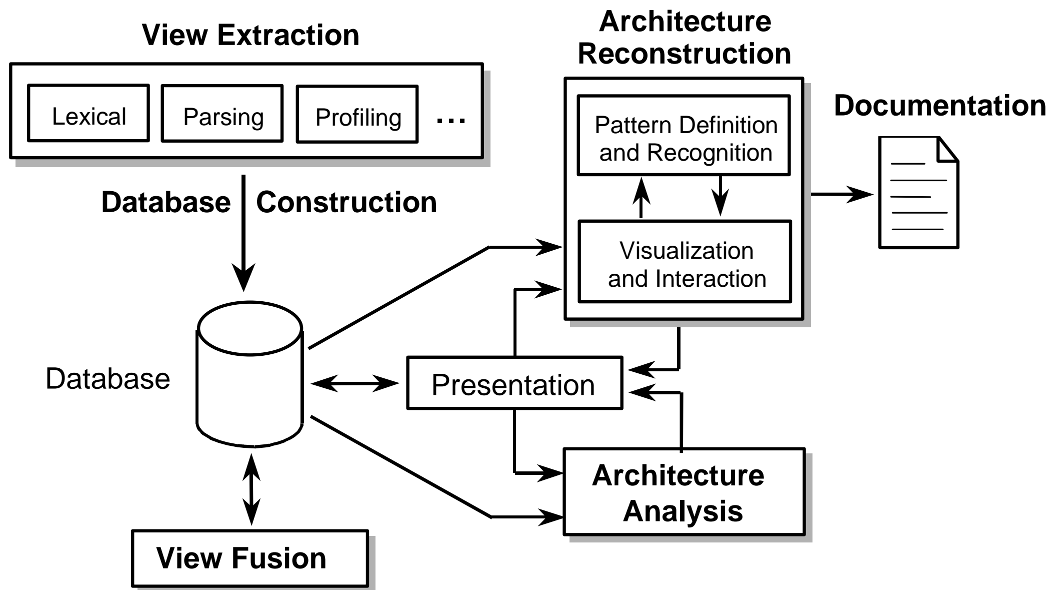

Figure 1: The Dali Workbench

2 View Extraction

In the View Extraction phase, we analyzed the existing design and implementation artifacts and constructed a system model based upon multiple views. We then used source artifacts (e.g., code, header files, build files) and other artifacts (e.g., execution traces) to identify and capture the elements of interest, define their relationships, and extract several fundamental views of the system.

The following sources of information were available:

- the source code of the systems
- feature specification for the motor systems. This was not particularly helpful in the reconstruction process.
- the staff who maintained the existing system

Both Imagix-4D[1] and SniFF+[2] tools were applied to extract views from the source code. After analyzing the output from both tools, we concluded that the Imagix-4D was more useful for extraction purposes. Table 1 shows the statistics for one of the motor systems.

Table 1: Source Statistics for One of the Systems

	Files	KLOC	Functions	Macros	Variables	Types
C/c	44	31	316	312	803	43
Header	23	7	0	1102	222	70
Total	67	38	316	1414	1025	113

While using Imagix-4D to carry out the extraction, we found it necessary to make minor changes to the code. These changes included

- the commenting of bit masks such as

  ```
  #define PRO_MASK 0b10000000
  ```
- compiler specific keywords such as `@tiny`, `@interrupt` had to be commentedassembler code between `@asm` and `@endasm` was commentedsome `#define` constructs for single bit accesses such as `#define b_motor_on status1._0` were changed to variable declarations so that they were picked up by the parser `unsigned char b_motor_on`

---

[1] http://www.imagix.com
[2] http://www.windriver.com/products/html/sniff.html

The code was parsed and loaded into Imagix-4D. The Imagix-4D tool exported information from its internal representation to a set of flat files. We then extracted the elements and relations that we required. Table 2 shows the set of elements and relations that were extracted from each system.

Table 2: The Set of Source Elements and Relations Extracted

Source Element	**Relation**	**Target Element**	**Description**
File	includes	File	a c preprocessor #include of one file by another
File	contains	Function	a definition of a function in a file
File	defines_var	Variable	a definition of a variable in a file
Function	calls	Function	a static function call
Function	access_read	Variable	a read access on a variable
Function	access_write	Variable	a write access on a variable

Since these systems run in an embedded environment without profiling tools, we were unable to use any standard dynamic information gathering techniques. We were also unable to instrument the code and get run-time output for the same reason. Instead, the architecture reconstruction was carried out using static information only.

3 Database Construction

After extracting the views, we converted the views into the Rigi Standard Format. The data was next converted into SQL code, then stored in a relational database. Several tools and techniques were incorporated into the Dali workbench to expedite this process. A perl script was applied to convert the elements and relations (Extracted View) file to Rigi Standard Format. The Rigi Standard Format files were then read by another perl script and output in a format that included the SQL code needed to build and populate the relational tables. Figure 2 describes this process.

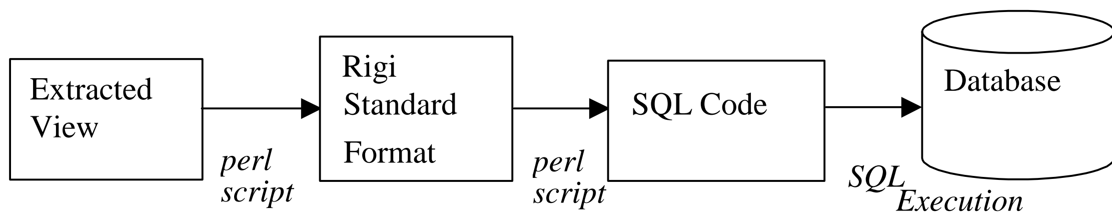

Figure 2: Conversion of the Extracted View to SQL Format

Dali currently uses the PostgreSQL relational database. By executing the SQL code in the file, one set of tables is created for each relation. The data is then entered into these tables. At that point, two additional tables are generated: *components* and *relationships*. The *components* table lists the set of source and target elements. The *relationships* table lists the set of relations extracted from the system.

4 View Fusion

The View Fusion process involves defining a set of queries to manipulate the extracted views and create *fused* views. We carried out two fusions in this case study:

1. Fusing the information from the various views to determine the types of elements in the *components* table.
2. Developing a very high-level system decomposition.

In some cases, a static call view may be fused with a dynamic call view. The reason is that a static view may not provide all of the architecturally relevant information required. In the case of late binding in the system, some function calls may not be identified until run-time, so there is a need to generate a dynamic call view. These two views needed to be reconciled and fused to produce the complete call graph for the system. In reconstructing these systems, however, only static information was available.

4.1 Updating the Components Table

The updated components table had the attributes *component name* and *component type*. The *component type* field for each component was not set when the table was first created. We set it by running a query on the table. An example query is shown in Figure 3. This query sets the type of each element in the component table to Function. The type of all components that are files, identified by having for example ".c" or ".o" in their file name, is set to File. In the last part of that query, all distinct variable names from the *defines_var* table were copied into a temporary table (tmp). The component table entry for these elements was updated by setting the type field to Variable.

```
=--  Make everything a function by default
UPDATE components
       SET tType='Function';

=--  Files: by naming convention
UPDATE components
       SET tType='File'
       WHERE tName LIKE '%.h' OR tName LIKE '%.H' OR tName LIKE '%.c'
       OR tName LIKE '%.s' OR tName LIKE '%.o' OR tName LIKE '%.inc'
       OR tName LIKE '%.C' OR tName LIKE '%.lib';

=-- Variables
DROP TABLE tmp;
SELECT DISTINCT variable
       INTO TABLE tmp
       FROM defines_var;
UPDATE components
       SET tType='GlobalVariable'
       WHERE tName=tmp.variable;

DROP TABLE tmp;
```

Figure 3: Example Fusion Query to Set the Type of Each Element

4.2 Decomposition into High-Level Components

One of the automotive motor systems contained code developed by the organization, external code for the CAN bus, and code to interface between these two pieces of software. In performing the reconstruction, we were interested in identifying the code in each of these parts of the system (high-level components). We wrote a query that created a separate table within PostgreSQL for each component and derived its visualization. Figure 4 presents a view of all three high-level components.

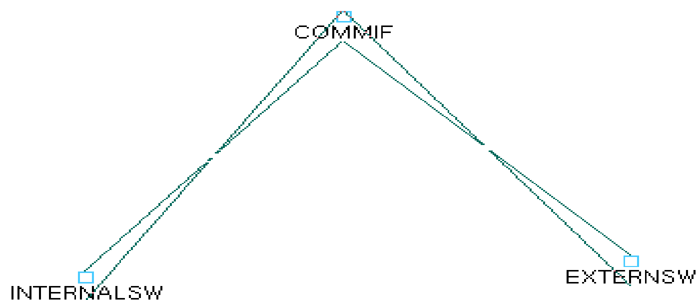

Figure 4: High-Level Component Decomposition for One Motor System

This high-level decomposition allowed us to concentrate our reconstruction efforts on the most important code to the organization, the INTERNALSW component.

5 Architecture Reconstruction

The Architecture Reconstruction phase consisted of two primary activity areas:

1. visualization and interaction
2. pattern definition and recognition

Once Dali has opened and loaded the database containing the various views, it generates a visual representation similar to that shown in Figure 5. We examined the visualization to define and recognize the various patterns, and uncover the architecture of the system.

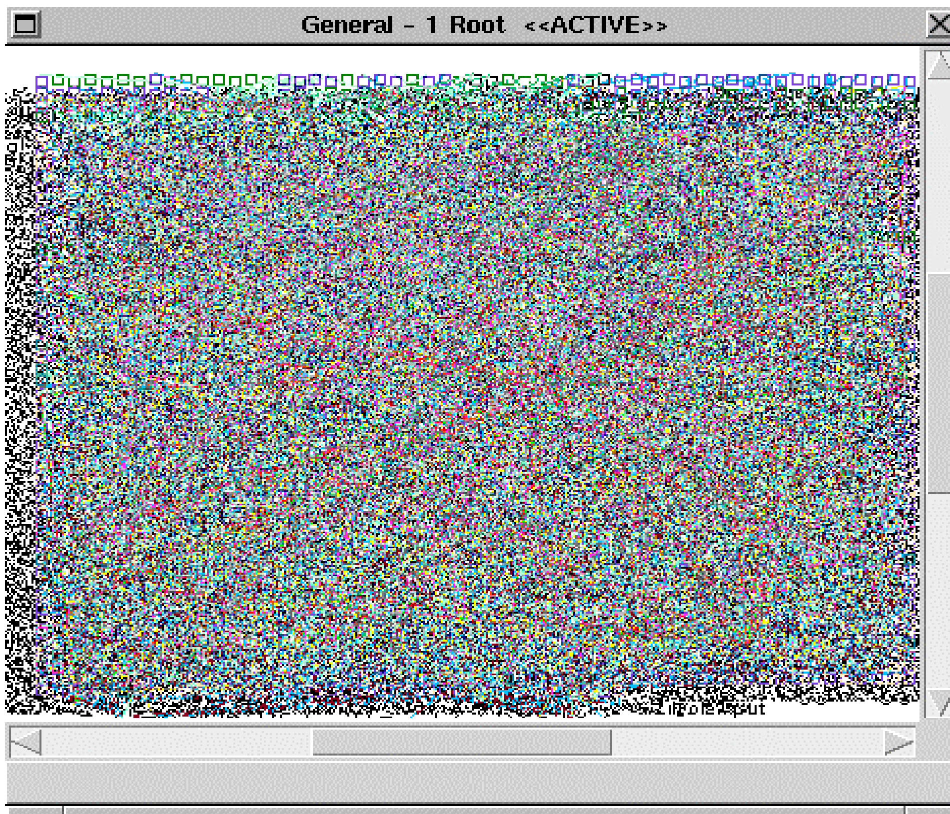

Figure 5: Initial Visualization Generated by Loading the View into Rigi

By aggregating the variables and functions the file contains, we were able to obtain a view of the system in terms of various files and their relations. Figure 6 shows this visualization.

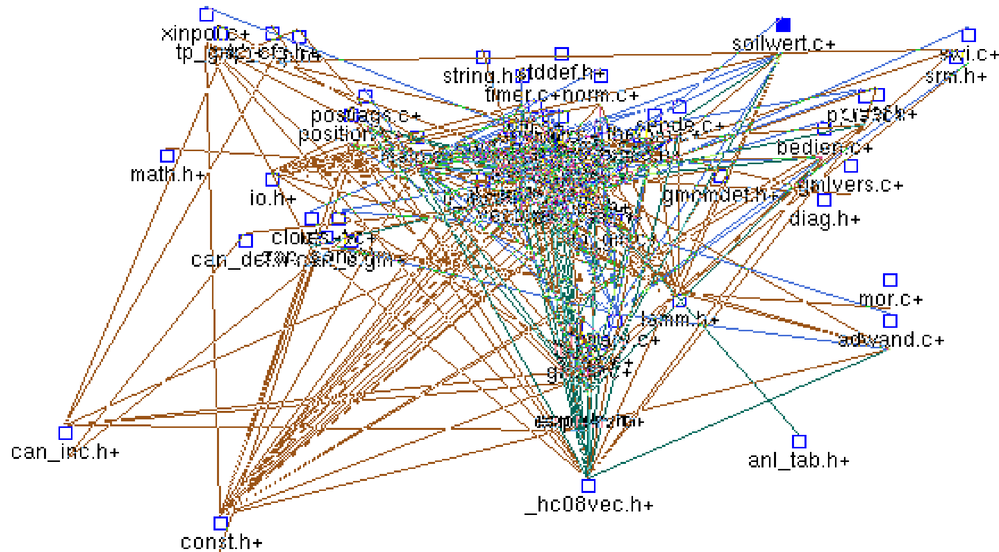

Figure 6: File Aggregation of Functions and Variables

In this view, the connections (arcs) between the nodes (files) represent different types of relations. For example, there may be simple relations such as one file includes another file, or the arc could represent composite relations of different types (a function in the file calls a function in the other, and a variable in the file is accessed in the other).

We began to identify architecture components by reading comments in the code to determine what functionality was being carried out in the file. We also read the documentation and talked to system maintainers. This enabled us to identify several components and determine which files belonged to a particular component. A visualization of the components was generated by creating and executing a pattern on the view shown above in Figure 6. The resulting view is shown in Figure 7.

By reading the comments and examining the code, we learned that that the HWPARAMETER component consisted of a set of declaration of hardware specific variables that were included in almost every file. We manipulated the visualization manually to hide this component. By similar means, we identified that the UTILITY component consisted of a set of utility functions that were accessed by almost all of the components. We manipulated the visualization to hide this component, as well.

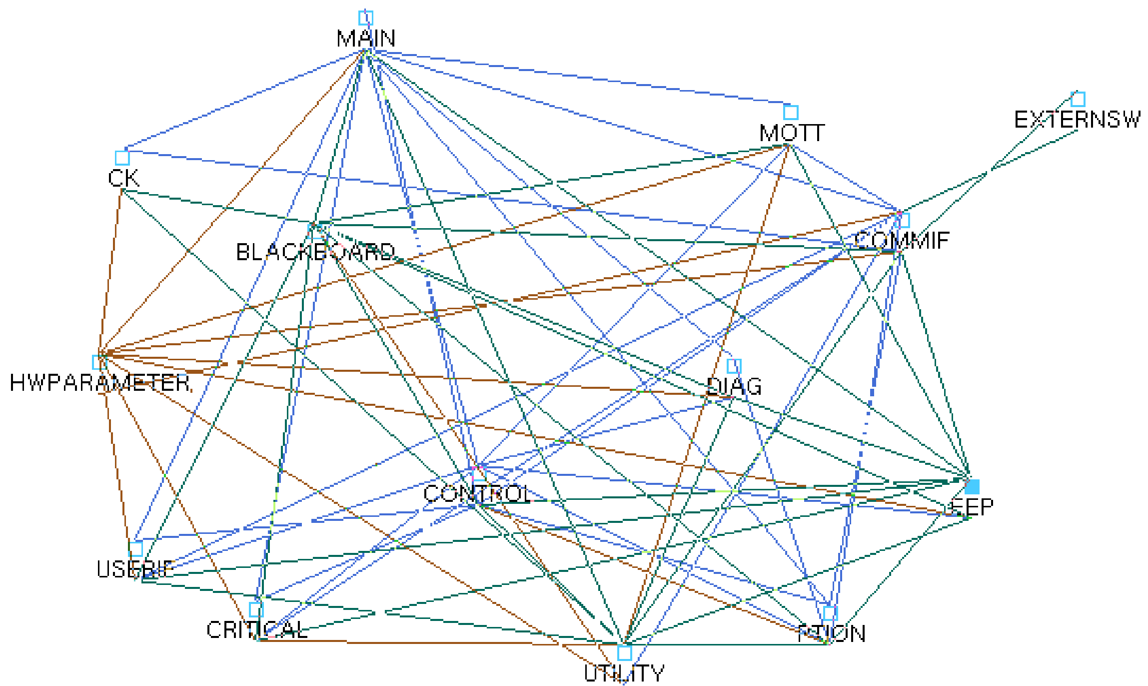

Figure 7: Components in the INTERNALSW

An examination of the BLACKBOARD component revealed how it was used. The component contains a set of files in which large sets of variables are declared. Other components access the values in these variables (access_read view) and some of the components set the values of these variables (access_write view). We identified this approach as the blackboard architectural style and produced the view shown in Figure 8.

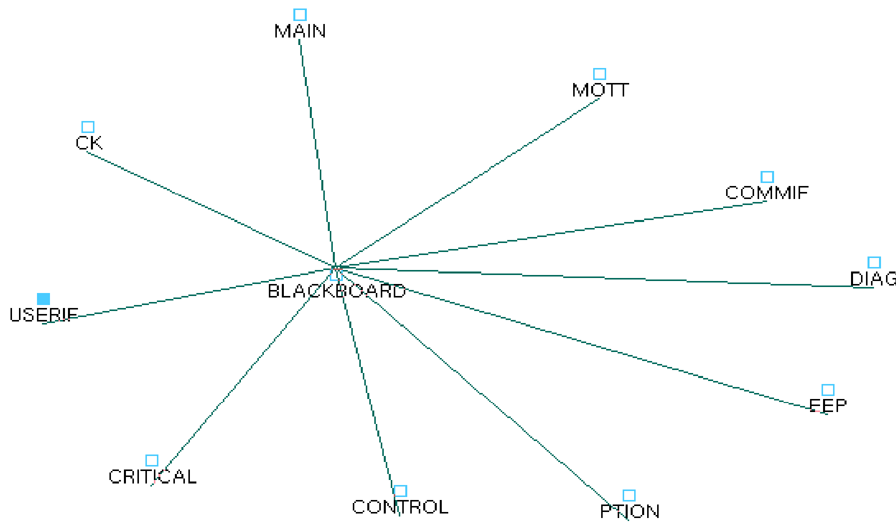

Figure 8: Use of the Blackboard Architectural Style

Next, we removed the BLACKBOARD component and the arcs to it to determine the functional decomposition of the system. Figure 9 shows a visualization of the functional decomposition. All the arcs except the calls view between functions in the components are hidden.

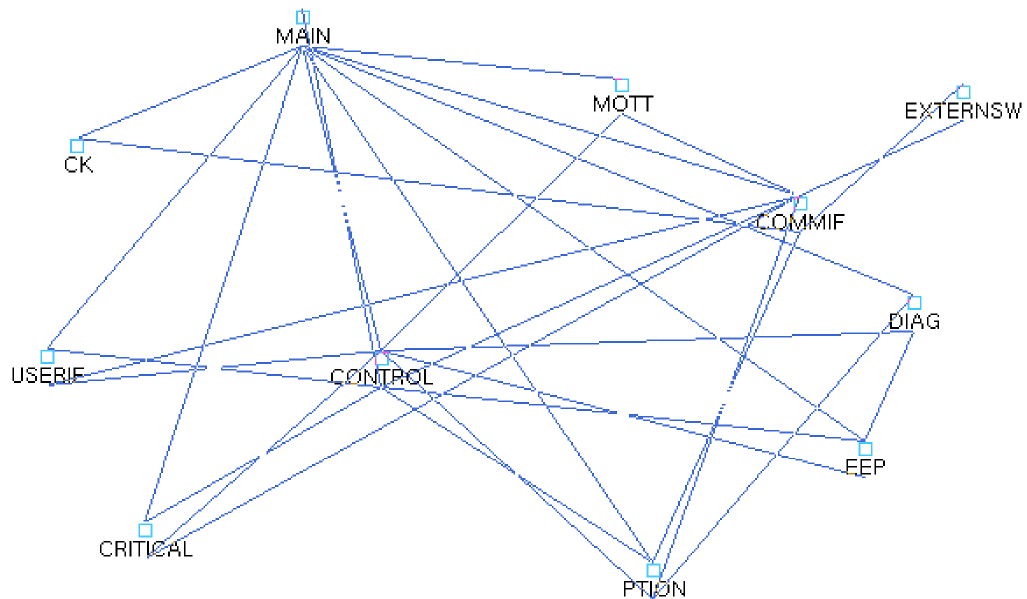

Figure 9: Functional Decomposition of the Components (with COMMIF and EXTERNALSW)

By removing the COMMIF and EXTERNALSW components from the view, we isolated software of interest in the case study. This updated view is shown in Figure 10.

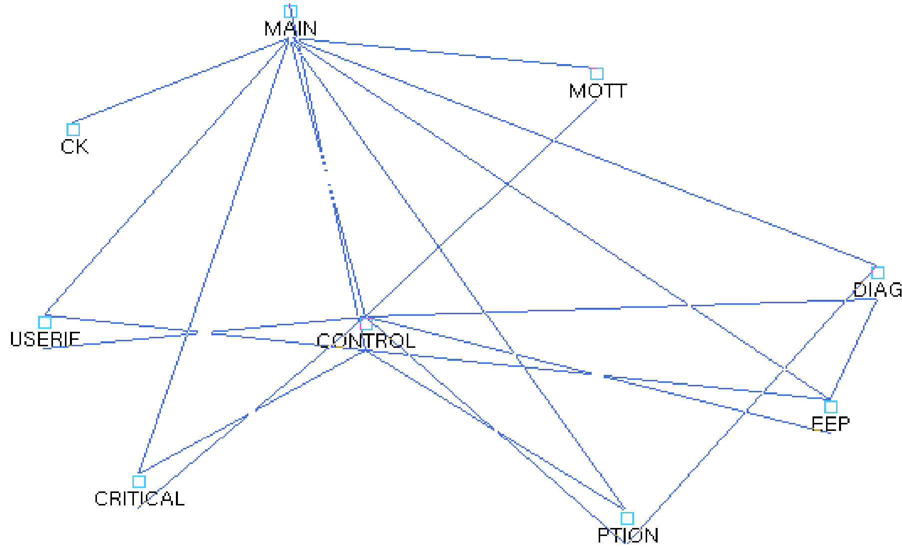

Figure 10: Functional Decomposition of INTERNALSW

In this view, we can identify some layering between the components. The MAIN component is at a higher layer than the CONTROL component and there is a layer under that containing the USERIF, CRITICAL, PTION, and EEP components. However this is not a strictly layered architecture, as we can see by the links between MAIN and the lower level components USERIF, CRITICAL, PTION, etc. By examining these links, we determined that the main function (contained in the MAIN component) contains a cyclic executive that calls various functions in the different components. This architecture style is typical of embedded system software with hard performance requirements.

6 Results

The view of the software in Figure 10 shows the various architecture components in one of the motor systems. The same process outlined in the last section was carried out for the three systems in the case study. In each of the systems, we identified similar components. We also came up with several important differences in characteristics among these systems by talking to the maintainers. For example, each system had different performance requirements. In the motor system with the CAN bus interface, certain information was obtained from the CAN bus. This information was not available in the other systems because they lacked this interface.

We further identified several architectural styles in the three systems:

- blackboard style. It shows that system functionality was divided across several computational steps; and each step is a knowledge source. Together, they follow a set of rules to form the solution. After each computation, several reactions are possible. We captured this using a large amount of state variables.
- cyclic executive style. It was identified in the MAIN component of each system. The MAIN component calls functions in each of the other components. This architecture style is typical of embedded system software with hard performance requirements and with both critical and less critical functionality.
- partial layered style. It was identified in each of the systems, although it was not strictly followed because of the cyclic executive.

We used the system reconstructions to determine the architectural and technical feasibility of developing a software product line from the three small automotive motor systems. A method, Mining Architectures for Product Line Evaluations (MAP) [Stoermer 01], has been developed that outlines the activities required to help stakeholders make that decision. Currently, the customer organization has an effort underway to develop a product line architecture for two of the motor systems and has produced a prototype product line. The organization is now investigating the feasibility of including the other system as well as additional products in that product line.

References

[Kazman 00] Kazman, R.; Klein, M. & Clements, P. "*ATAM: Method for Architecture Evaluation*" (CMU/SEI-2000-TR-004 ADA 382629), Pittsburgh, PA: Carnegie Mellon University. WWW. URL: <http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html> (2000).

[Müller 93] Müller, H. A.; Mehmet, O. A.; Tilley, S. R. & Uhl, J. S. "A Reverse Engineering Approach to System Identification." *Journal of Software Maintenance: Research and Practice 5*, 4 (December 1993): 181-204

[Stoermer 01] Stoermer, C. & O'Brien, L. "MAP: Mining Architectures for Product Line Evaluations." *Working IEEE/IFIP Conference on Software Architecture*, Amsterdam, The Netherlands, August 28-31, 2001.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	July 2001	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Architecture Reconstruction to Support a Product Line Effort: Case Study	F19628-00-C-0003

6. AUTHOR(S)

Liam O'Brien

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2001-TN-015

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Recently, technical staff members of the SEI performed architecture reconstructions on three small automotive motor systems. One system has an interface to an external bus within the automobile (the Controller Area Network [CAN] bus). The other systems do not have this interface. This technical note describes the architecture reconstruction process that was followed. It provides an overview of the Dali workbench developed to support this process, presents the various activities in each phase of the process, and outlines the results that were produced.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Architectural styles, architecture reconstruction, reverse engineering	27

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL