A Comparison of Four Software Architecture Reconstruction Toolkits

Meena Jha[1], Piyush Maheshwari[1,2], and Thi Khoi Anh Phan[3]

[1]School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia;

{meenaj, piyush}@cse.unsw.edu.au

[2]National ICT Australia Ltd;

[3]Faculty of Engineering, University of Technology Sydney, Australia;

Thi.K.Phan@uts.edu.au

UNSW-CSE-TR- 0435

October 2004

Abstract

This report discusses the evaluation of four software architecture reconstruction tools. This evaluation is needed because many legacy system needs to be reconstructed as the requirements change for the purpose of modernization. *Software reconstruction* is a tool-based iterative and interpretive process. Software architecture reconstruction tools support software engineers in the process of recovering the "as-built" architecture of an implemented system. The tools extract information about the system and aid in building and aggregating successive levels of abstraction. If the tools are successful, the end result is an architectural representation aids in reasoning about the system. There are several commercial reconstruction tools on the market providing different capabilities and supporting specific source code languages. In this report, we evaluate four software architecture reconstruction tools on the criteria of extraction ability, abstraction ability, navigation ability, ease-of-use, views, completeness and extensibility. The tools presented for evaluation are Dali workbench, PBS, SWAG Kit and Bauhaus. Capabilities of these tools are evaluated by applying them to medium size software *'concepts'*.

1. Introduction

If architectural reasoning is performed at the initial stage of a system's lifecycle, it will facilitate the detection of any non-conformance with respect to design requirements. Studies show that between 50% and 90% of software maintenance involves the understanding of the software being maintained [27]. During maintenance, modifications may occur and therefore may have an impact on the software system architecture. Software architects have to understand, analyze, and reason about the as-built software architecture of a system to modernize it [28]. There will be a need for architecture reconstruction if the existing architecture is not yet well understood. Modernization may occur when there is a business need to upgrade software. Most of the time the software documentation of legacy systems do not represent existing system or they are out of sync with actual systems due to poor documentation during maintenance process [9]. Therefore, recovering or reconstructing system structural information is an essential process for software maintenance and evolutionary development.

Software architecture reconstruction is infeasible manually when working with medium or large size software systems because of their complexity. As a result, tools are necessary for the automation of architectural reconstruction process. However, no tool is fully automatic since it requires human interaction at some stage to obtain application-specific knowledge about the system [3]. In this report, four software architecture reconstruction tools which are commonly used and well known in the architecture reconstruction research area are thoroughly examined to assess and compare their capabilities. The case study has adapted a three-step process in software architecture reconstruction: extraction of architectural information, abstraction of extracted data and visualization of abstracted modules. The tools' capabilities are evaluated in terms of extraction ability, abstraction ability, navigation, ease-of-use, views, language support, extensibility and completeness.

The report is structured as follows. Section 2 presents an overview of the four selected toolkits. Section 3 describes the case study. Section 4 describes the functionalities of the toolkits. Section 5 analyses and compares the toolkits. Finally the conclusions and future work, in section 6, summarize the strengths and weaknesses of each of the tool sets.

2. Toolkit Overview

There are a number of tools available that support the parsing, extracting source code and analyzing software system's architecture. Four tools are selected in this study. These are:

1. *Dali* workbench [13]
2. *PBS* toolkit [6]
3. *SWAG toolkit* [17], and
4. *Bauhaus* [10]

The criteria for a selection of a tool is not only being able to extract information from source code but also being able to abstract and visualize system components to a higher level. Other tools such as *Sniff+*, *Understand* and *Imagix 4D* are not categorized as software architecture reconstruction tools since they do not support the abstraction and/or visualization of the software architecture [8, 15, 19]. Nevertheless, they can be used as external extractors to produce formatted outputs which then can be inputted into one of the examined software architecture reconstruction tools. Studies have been performed to evaluate reverse engineering tools for program understanding [3].

2.1Configuration of Tools

Table 1 gives descriptions of the languages, platform supported by the tools, their fundamental components and some of their development history

		PBS and *SWAG*		
Features	***DALI***	***PBS***	***SWAG***	***Bauhaus***
Supported platforms	Linux	Windows and Linux	Linux	Linux and Windows
Installation platform	Linux Redhat 9	Windows XP	Linux Redhat 9	Linux Redhat 9
Developed by	SEI	University of Waterloo, Canada		University of Stuttgart, Germany
Space occupied	7Mb	15Mb	98Mb	48Mb
Supported language	C, C++, Java and Fortran	C	C and C++	C and C++
Components	- *Postgres SQL* 7.0+ - *Rigi* - Tcl/tk scripts - *csh* shell - *Perl* scripts	- Web server - *csh* shell - C language parser *cfx* - Manipulation tool *Grok* - Java-based user interface *lsedit*	- *cppx* - *Grok* - *lsedit*	- C frontend *cafe* - Preprocessor *cafeCPP* - Frontend *cafeCC* - Linker *imllink* - Script *iml2rfg*
Required external software		Java Virtual Machine (JVM)	JVM	- *emacs*
First release	1998		Sep 2002	2000
Latest update	2001	1997	Feb 2003	Dec 2003
Taken over by	ARMIN[1]	*SWAG*	N/A	N/A
Input format	Rigi Standard Form .rsf files	- Source code - contain.rsf which specifies the architecture of the system (optional)		- Source code - RSF - GXL
Output	- Hierarchical graphical representation of the highest level architecture. - Graphical representation of the conformance to designed architecture.	Graphical layouts of the main software system and its subsystems.		- Hierarchical graphical layouts of the main software system and its subsystems. - Various types of views including call graph, file view, logical view, type view and user defined view
Previous big case study	- Nokia [14]	Linux operating system [6]	Linux and VIM editor [17]	- Web Browser Mosaic and Chimera [11]

Table 1: Configurations of tools

2.2 Dali Workbench

Dali is an open lightweight workbench which utilizes Postgres SQL database, Perl scripts and *Rigi* Interface. It accepts well-formatted data as input, populates the input to tables in a database. By manipulating the database, the architectural information can be grouped to show different abstract levels of the system's architecture, which are presented in a graphical interface [13].

2.3 The *PBS*

PBS was developed as a Software Bookshelf which is a web-based model for the presentation and navigation of software systems' architectural information [7]. However, a new reverse engineering tool named SWAG Kit has been developed to enhance the capabilities of PBS. SWAG is used to generate software landscapes from source code through three phases namely extraction by cppx tool; manipulation by prep, linkplus and layoutplus tools and presentation by lsedit tool [7].

2.4 Bauhaus tool

*Bauhaus t*oolkit is used for architecture recovery as well as for program understanding and code auditing during maintenance process [2]. This toolkit has a set of tools to extract, analyze, query and visualize information about existing software.

3. Overview of '*concepts*' Program

'Concepts' calculates a lattice of concepts from a binary relation and produces output in a format specified by the user. Concept lattices are also known as Galois lattices. Any set of objects can share a (possibly empty) set of common attributes. The same for attribute sets which share common objects. Therefore, every set of objects determines a set of common attributes and every set of attributes determines a set of common objects, forming a pair of object and attribute set. A pair (O, A) of such two sets is called a 'concepts', if the following holds: the set of attributes common to the objects in O is A and the set of objects commonly shared by attributes in A is O [10]. A metrics summary of 'concepts' which was obtained from "Understand for C" is:

Files:	31
Functions:	125
Lines:	9381
Lines code:	3861
Lines comments:	2603

The only documentation about 'concepts' program available is an installation guide and a brief description about the software. No design or architectural documents exist. Understand for C and Imagix 4D have been used in this case study for the extraction of 'concepts' source code.

4. Toolkits Functionalities

4.1 Dali Workbench

View extraction: View extraction is the process of gathering and analysing existing design and implementation of artifacts such as source code, architectural or design documentation. The Dali workbench does not have any extraction tool; thus, it is the users' responsibility to extract structural information from the source code. There are a number of tools available on the market to choose from such as Imagix 4D, Understand and Sniff+. In this case study, Understand for *C* is used for source code extraction. *Perl* scripts were written to transfer data from reports generated by Understand for *C* to RSF which is the only format that Dali accepts. The schema for the C-written 'concepts' program:

function	*calls*	function
file	*defines_fn*	function
file	*defines_global*	global_variable
file	*defines_macro*	macro
file	*defines_ADT*	ADT[2]
file	*uses_macro*	macro
function	*uses_global*	global_variable
function	*uses_ADT*	ADT
function	*defines_var*	local_variable

The Understand software can only extract static information, dynamic state of the 'concepts' program can be captured by using other tools or facilities like profiling or from makefile..

Database Construction*:* After obtaining the view(s) of a system, the views can be stored in a relational database. PostgresSQL 7.0 is used in the case study.

View Fusion: If more than one view can be obtained from the view extraction stage, those views can be combined to create a fused view [13]. Fusions are defined by using SQL. An SQL file name fuse-types.sql was written to specify entity types for the views extracted in 'concepts'. The quality of static view can be improved by combining with dynamic view.

***Architecture Reconstruction*:**

Dali uses Rigi as its interface. Figures 1 and 2 are screenshots presented when rigiedit is started and is loading a new database to the main Rigi interface window.

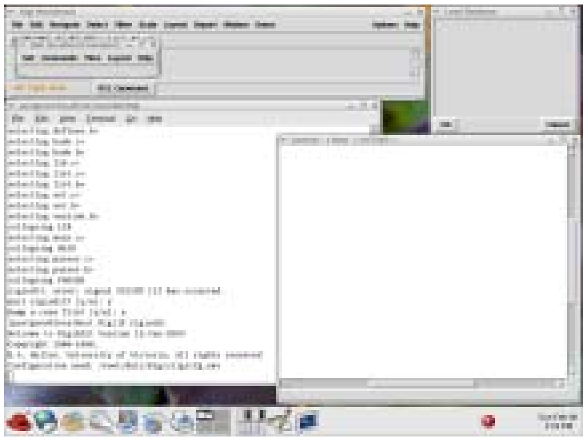

Figure 1: Dali interface

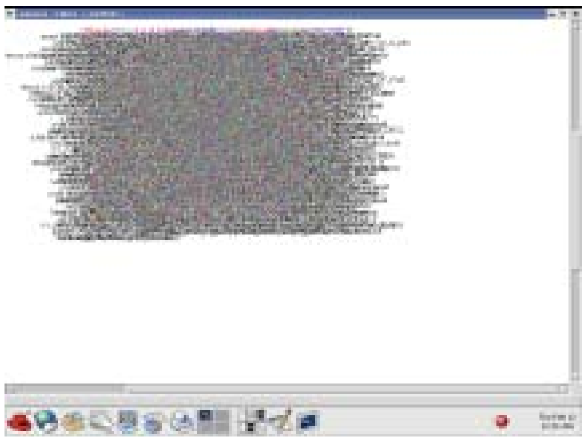

Figure 2: Raw concrete module of '*concepts*'

Dali allows user to construct more abstract views of a software system from more detailed ones by developing aggregations of elements. This capability is achieved by applying patterns which are defined as a combination of SQL queries and Perl scripts. The SQL query identifies nodes in the repository which are grouped to form a new aggregation; the Perl command is used to transform names and perform some manipulations of the query's results. There were three patterns applied in the 'concepts' case study.

1. The first one was function aggregation which aggregates functions and their local variables
2. The second pattern was file aggregation which accumulates all elements that were defined in files such as aggregated functions, global variables, macros and ADTs.
3. The third one was concepts-architecture aggregation which groups a number of related files into a module.

The first two patterns were application-independent patterns since they leverage architectural information common to many applications. The last one would be dependent on domain knowledge of the system; this is where human interaction is required during the reconstruction process. Based on the decomposition of sub-systems specified in the conceptual architecture, the concepts-architecture pattern was developed. Figures 3, 4, and 5 display the graphical layouts of ‘concepts’ architecture after each pattern was applied sequentially.

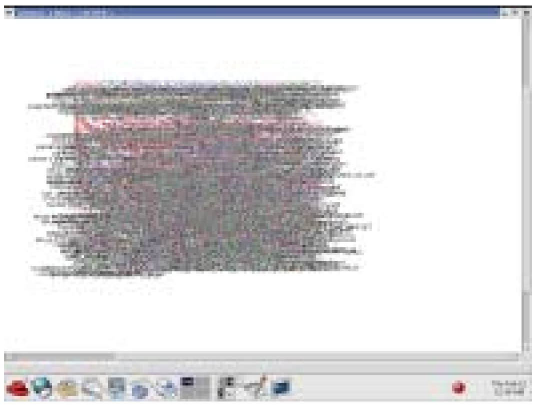

Figure 3: The ‘concepts’ architecture, file patterns were applied

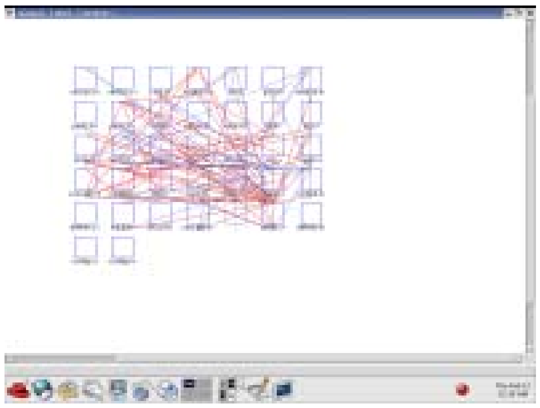

Figure 4*:* The ‘concepts’ architecture, after function pattern was applied

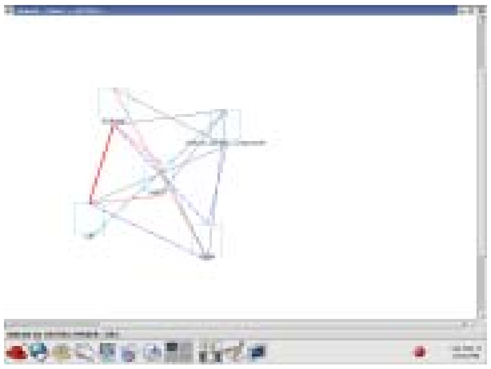

Figure 5: The ‘concepts’ architecture after function, file and ‘concept’-arch patterns were applied.

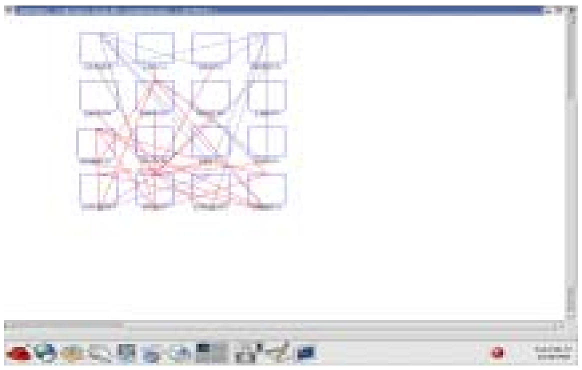

Figure 6: domain-specific-components

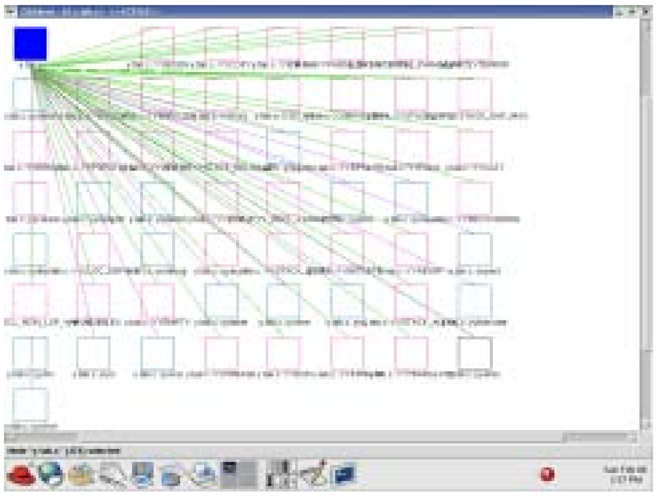

Figure 7: The content of y.tab.c+ node in the domain-specific-components sub-system

Architecture Analysis:. Using Rigiedit interface tool, the as-designed architecture of ‘concepts’ is drawn. When both as-implemented and as-designed architectures of a program are available, a conformance analysis can be performed by using RMTool [25].

4.2 PBS and SWAG Tool

The Portable Bookshelf (PBS) is a toolkit used for generating a ‘software bookshelf’ [7].A software bookshelf for a large system can provide an easily accessible Web-based structure for storing information about a system. The information provided in bookshelf includes source code, as well as other documentation about the system.

Reconstruction using PBS**:** Similar to Dali, PBS uses RSF format to describe facts. PBS has a C extractor to extract input source code directly. The results retrieved from the extractor then can be manipulated by using a fact manipulator. The fact manipulator provided by PBS is Grok which is a software tool operates on facts written in RSF [7]. Also, PBS has a Layouter tool which reads facts representing a graph and adds layout attributes to it. Finally, the graphs are viewed by a Landscape Viewer which is a Java applet. To obtain architecture of a software system, a file in RSF format needs to be written to specify the containment structure of the system. PBS can automatically generate such a file. When PBS was evaluated its functionalities were not fully supported and this tool has been taken over by SWAGKIT.

Reconstruction using SWAGKIT**:** SWAGKIT interface is similar to that of PBS; however diagrams do not get displayed on web pages. The following steps were involved in the reconstruction process of ‘concepts’ [17]:

- Extract facts from source code by using cppx tool which produced *.ta file from the original source files
- Prepare the facts by using prep, which produced *.o.ta from the extracted facts.
- Link the facts by using linkplus which produced out.ln.ta from *.o.ta.
- Layout the facts by using layoutplus which produced out.ls.ta from out.ln.ta
- Finally, the graphs can be visualised by lsedit.

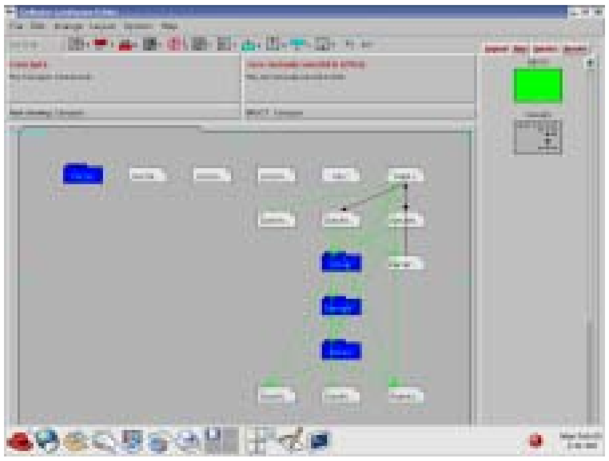

Figure 8: Highest level Architecture of concepts retrieved from SWAGKIT

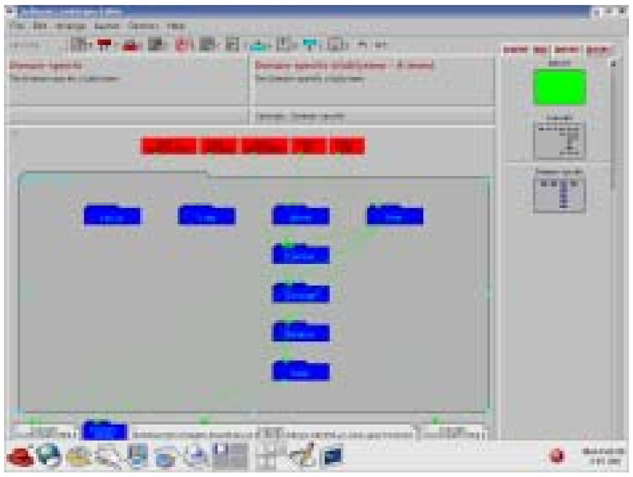

Figure 9: One of the main sub-systems of 'concepts' Architecture

4.3 Bauhaus Toolkit

Bauhaus provides a variety of tools to assist the maintenance of software and recovery of software's architecture [2]. Bauhaus uses three special tools to analyse source code:

- A C analyser front end cafe which generates intermediate language IML from the source code.
- A linker imllink which performs global name resolution on IML files and generates a globally linked IML that describes the whole system.
- A script cafeCC which acts as a front end to cafe and imllink

The 'concepts' program needs to be compiled with cafeCC.

***Tools support maintenance*:**

1. *Measure code attribute:* The Bauhaus toolkit can measure a number of source code attributes including lines of code per function, cyclomatic complexity and maximal nesting [2].

2. *Detect code duplication:* The Bauhaus toolkit allows user to discover clone codes automatically, it even can distinguish three different kinds of code clones: copies that are identical (red colour), copies that are structurally identical (blue colour), and copies where statements were added or removed (green colour).
3. *Detect dead code:* The Bauhaus toolkit offers a tool called iml_dead_functions to detect dead function and the results retrieved from this tool can be browsed by emacs

Reconstruction using Bauhaus:

1. ***Extraction***

Bauhaus extracts following types of global declarations:

- routines: C functions defined in source code and included from the environment.
- user-defined types: such as structs, unions, enums and typedefs.
- objects: global variables and constants.
- members: record components of structs and unions.
- indirect calls: calls through function pointers

Bauhaus also captures the aggregation of global declarations in modules and the containment of modules in directories. The relations between global elements can be lifted to the modules in which these global elements are grouped and finally to directories or sub-systems. This feature is supported by setting the option _total_lift in the iml2rfg tool which reads the IML file and creates a graph called resource flow graph with all global declarations. Bauhaus isolates global declarations.

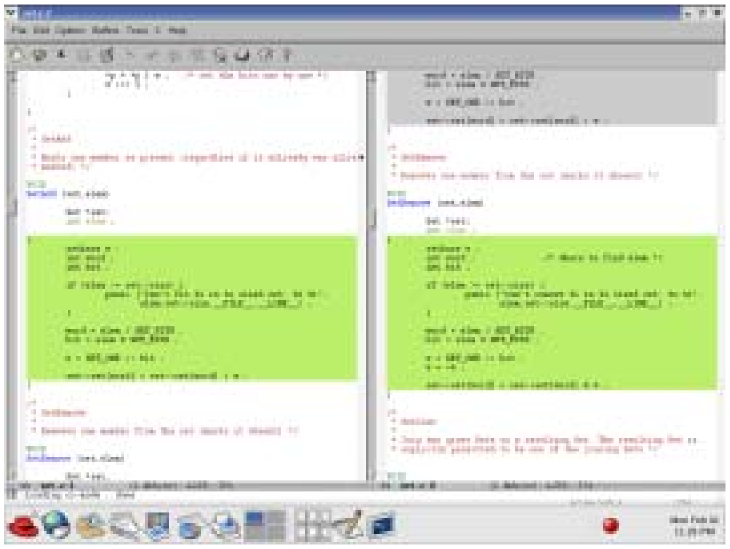
Figure 10: Code duplication of type 3.

2. ***Visualisation of dependencies***

Bauhaus interface has two main windows: the workbench which contains the main menu and displays status information and the view box which helps users manipulate various views of the graph. Components can be browsed from top to bottom of the hierarchical architecture in a view mode of user's choice. There are three kinds of views:

- Flat: All nodes and edges are shown in a view.
- Hierarchical: only the top-level nodes are shown.
- Shrimp: the level arcs (arcs that are connecting two nodes on different hierarchical levels) can be displayed.

Figure 11 shows the nodes and edges of two main 'concepts' sub-systems in Shrimp view.

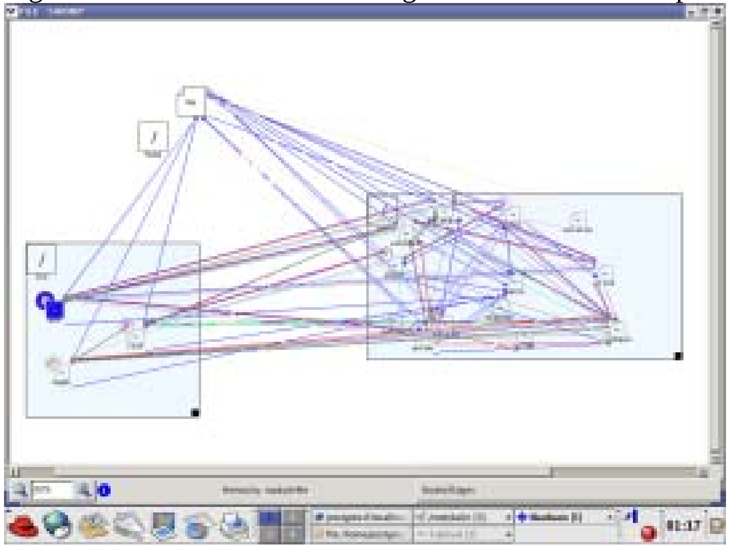

Figure 11: Nodes within two subsystems lib and src and dependencies between them.

The nodes' information can be analysed choosing Imports, Exports, Clients and Details view.

- **Imports** are those entities *C* uses from other components.
- **Exports** are those entities of *C* that are used by other components.
- **Clients** are those components that use entities of *C*.
- **Details** are those entities of *C* that are not used by other components.

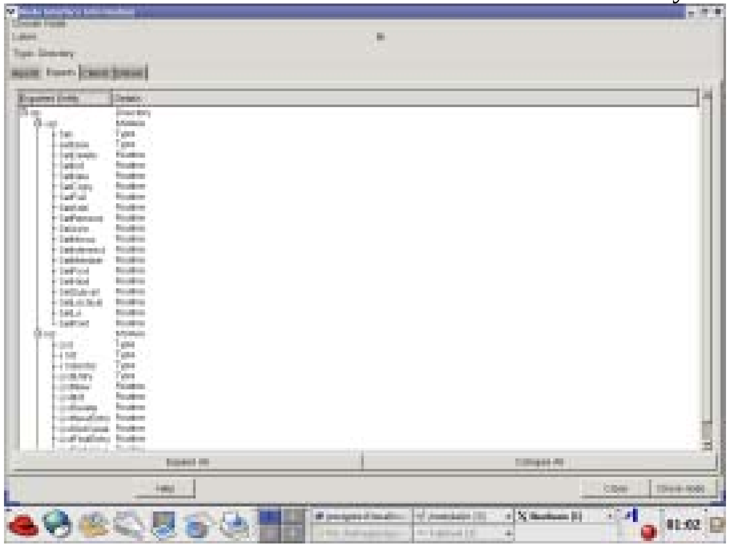

Figure 12: Node Information

Different layout algorithms can be applied to the graphs. Some common ones are:

- Circular layout: nodes are displayed circularly.
- Grid: nodes are displayed in a grid.
- Stretch: nodes can be moved apart in X and/or Y axis.
- Spring layout
- Tree overviews
- Sugiyama
- Tighten: Close ranks nodes in X and/or Y axis.

Transitive dependencies among components

Bauhaus also allows users to analyse one specific component and its dependencies to other components. All routines can be specified that are needed to be executed until function ListInit is executed. Figure 13 depicts the result of the process:

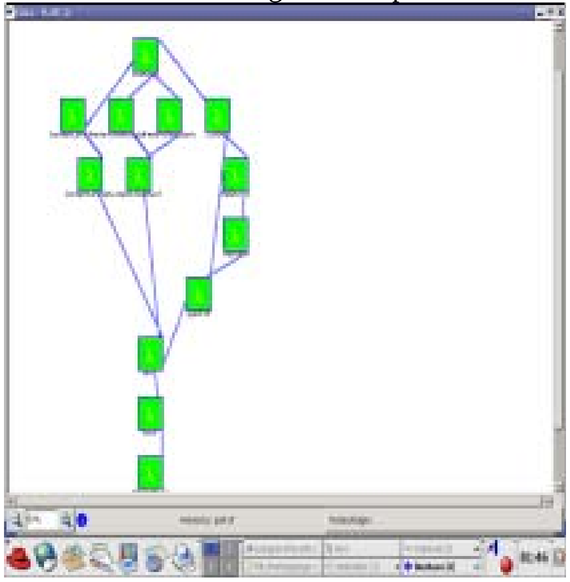

Figure 13: Hierarchical call for ListInit routine

3. ***Architecture conformance analysis***

Bauhaus allows users to check the conformance of a system's implementation if there is a specification of that system's intended architecture. Bauhaus lets users specify the high-level architecture and map the concrete components onto the architecture, then compares the high-level architecture to the concrete components and their dependencies. [2].

The output of the comparison is a reflexion model highlighting:

- **Convergences**: are references in the hypothesized model also present in the concrete model.
- **Divergences:** are references in the concrete model for which no reference in the hypothesized model exists.
- **Absences**: are references in the hypothesized model not present in the concrete model.

5. Analysis of Tools

5.1 Extraction capabilities

Integrated extraction tools: PBS, SWAGKIT and Bauhaus have an extractor included in their toolkits. Extractor is not part of Dali workbench, and as such source code has to be extracted by external tools such as Sniff+, Imagix 4D and Understand. PBS is only able to parse C code, while SWAGKIT can parse both C and C++ programming languages by the use of cppx front end analyser. Since the extraction process can be done outside the Dali workbench, Dali can be applied to programs written in more variety of languages as far as the input file is in RSF format. Bauhaus parsing capability is more extended since it has analysers for ANSI-conformant C, C++, Java and COBOL as well as accepts inputs in RSF and GXL format. Also, most of not strict ANSI-C programs can be processed by the Bauhaus analysers through providing appropriate macros [2].

Hiding system library calls: For SWAG toolkit, the isolation of standard library files is not done automatically. The users can group those files into a sub-module by modifying the arch-contain.rsf file. The arch-contain.rsf file identifies how the system should be decomposed of in a tree hierarchy. Similar process has to be done in PBS to achieve this goal; otherwise a Grok script has to be written to remove those library files from the results. Bauhaus toolkit has an option to identify which library calls can be removed from the analysis.

Parsing capability: In Bauhaus a set of types extracted is flexible as it allows users to define their own schema for the data; while in Dali the extraction is dependent on which external extraction tool was chosen and the architect who performs the task. Local variables were not extracted in Bauhaus. Dali might provide more flexibility for users because the users can decide how deep the architectural level they want to obtain from the analysis. A correct compilation of the source code is required for PBS extractor to work.

Ease of Use: All of the integrated extraction tools are easy to use since each provides straightforward scripts which require little human interaction.

5.2 Abstraction Capabilities

All four toolkits can abstract low-level information to high-level representation through the composition of models. The abstraction of data can be done automatically according to files-modules-directories hierarchy in Bauhaus while it is not automatic in Dali and PBS. Abstraction in PBS and SWAGKIT is achieved by the Grok tool. There are many scripts provided with PBS that automate the execution of abstraction. However, a pre-defined sub-system structure has to be written manually before the scripts can be performed.

The abstraction in PBS is fast and is done before the visualisation therefore it is more time efficient. It is also possible for users to write their own Grok scripts to manipulate the data on a specific purpose such as analysing only a portion of a system or using an external extraction tool. Dali populates low-level data extracted from an external tool into a Postgres SQL data repository by using Perl scripts. Dali uses Rigi tool to provide interface and abstraction ability. Manual inspection of the nodes displayed in the resulted graph reveals that all system library calls are not included; they have been deleted during the abstraction process. The users have to write a pattern query file which is a combination of a SQL query and a Perl script command for each abstraction. A number of sequential queries can be grouped together to from a query set which can be executed in one time and save time for users. The first abstraction process is quite simple, but higher-level sub-system decomposition normally requires domain knowledge. In Bauhaus, the abstraction is done automatically. During the extraction, a resource flow graph was created to represent all global declarations (nodes) and their dependencies (edges). When this graph is visualised by the Bauhaus graph editor, the components of the system can be browsed from top (directories) to bottom (source code). In addition, logical abstraction can be performed via a component mining process.

The base view of the component mining contains the global declarations and their relationships. The base view is automatically derived from source code. The user view records the information contributed by the user are components that have been detected and confirmed. Among these steps, the user monitors the detection process by selecting analyses and adjusting their parameters and by validating the candidates proposed by the automatic techniques. The Bauhaus software is responsible for automatic analyses, computation of the metrics for the proposed candidates, presentation of the results and keeping records of the user decisions. The user selects an analysis to be applied. “The analysis takes into consideration the components that were confirmed by the user (in the first iteration there are none). Thus, the analyses are applied incrementally; that is, they may cluster only those global declarations that have not been clustered before to new or existing components. Then, the candidates are presented to the user for acceptance. The user validates the candidates and the accepted components enter the component memory—that is, the user view. In each iteration, the user selects and combines different analyses to find components that could not be found by previous analyses. The process ends when the found components are sufficient for the task at hand or no further component can be found anymore. Several analyses

can be selected and applied in parallel. Then, the intersection, union, and differences of these analyses can automatically be ascertained and the user can investigate and validate them." [2]
Figure 14 shows the result view for component mining process where components are identified based on abstract data type heuristic or internal access heuristic.

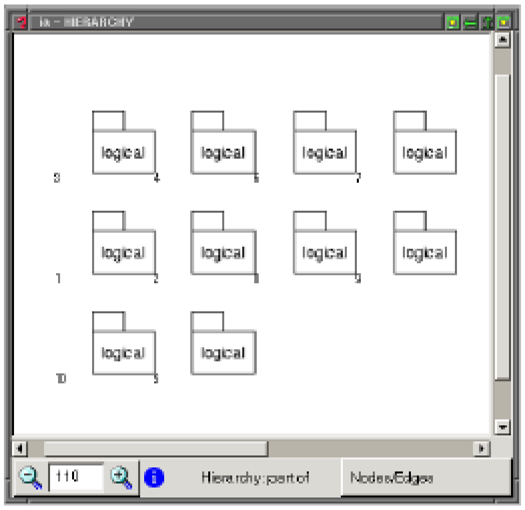

Figure 14: Result view for component mining

5.3 Visualization Capabilities

The features table 2 describes are the capabilities of the visualisation tools. A check mark ✓ in a specific column indicates the presence of the corresponding feature on that row.

	Dali	PBS	SWAG	Bauhaus
Node type in shape	-	-	✓	✓
Node type in color	✓	✓	✓	✓
Node type in text	-	-	-	✓
Edge type in color	✓	✓	✓	✓
Edge type in text	-	-	-	✓
Move nodes and arcs	✓	✓	✓	✓
Node hierarchies	✓	-	✓	✓
Resize nodes	-	✓	✓	-
Bi-directional edges	-	✓	✓	-
Scroll	✓	-	-	✓
Zoom	✓	-	-	✓
Annotation	✓	✓	✓	✓
Different Layouts	✓	-	✓	✓
Saving graph	✓	-	✓	✓
Opening graph	✓	-	✓	✓
Create user's own graphs	✓	-	-	✓

Table 2: Feature table lists visualisation capabilities of the tools.

All tools use graphs that contain only nodes and edges to present architectural information. Each node represents a component in the system while each arrow represents the relationship between components. The Dali workbench and the PBS bookshelf use only colour code to distinguish different types of nodes and edges. Bauhaus and SWAGKIT utilise both colours and shape to differentiate node types and only colour for edges types. These two tools also have a panel describing information of all node and edge types. Moreover, SWAG has a map which is a miniature of the content structure at each hierarchical level of the target node.
PBS and SWAG graph viewers do not support scrolling while Dali and Bauhaus do. In addition, Bauhaus provides zoom features and Dali offers scale feature which is quite similar to zooming but less flexible. Also, only Bauhaus offers multiple views of graphs
Dali uses uni-directional arcs. To represent the relations between any two components Dali uses two uni-directional arcs. Dali allows users to create their own diagrams which represent the architecture of a system according to design documentation and then use conformance analysis tool to compare between the as-designed architecture and as-implemented architecture. Similar feature is implemented in Bauhaus. In addition, both tools have specific colour code to distinguish between the divergence, convergence and absence.
Navigation features: All the tools except PBS support hierarchical browsing which reinforces the abstraction of components into modules. Multiple nodes can be collapsed together into one parent node. The children nodes can also be viewed. While Dali and Bauhaus open a new window to display children nodes, SWAG and PBS display a new graph in the same main window (the current one is cleared). Bauhaus can either display a nested graph in the same window or display children nodes in a new window. Bauhaus has a manager to control these windows.
Views and Layouts: The readability of a graph can be examined by analysing the issues of node shape, node size, and graph layout, number of arcs crossing and placements of nodes. In Dali nodes are represented as a square box whereas in PBS it is a rectangle box. SWAG and Bauhaus supply a number of different shapes for nodes such as circle, square, hexagon and triangle. Dali and Bauhaus allows full editing of the graphs including adding, deleting, modifying and moving nodes.
PBS provide only one option of layout which is automatically invoked upon when the user interface is started while Dali, SWAG and Bauhaus have more options of layout techniques. However, Dali and SWAG does not have as many layout methods as Bauhaus does. In Dali, an appropriate graph layout is not automatically invoked. Horizontal and Vertical layouts gives tight structure of nodes and their names. Bauhaus toolkit has a number of layout options to choose from depending on type of graph needed to be presented. All the tools allow arcs crossing and arcs can even pass through nodes to which they are not attached. This can make graphs look complex.

5.4 Other attributes

Language support: All tools support the parsing of C language. However, many existing legacy system were written in other languages. Dali supports C, C++, Java and Fortran.

Completeness: Functionalities that a software architecture reconstruction should have are extraction ability, abstraction ability, views, and navigation and conformance analysis. Bauhaus has tools that support all these functionalities. Dali also possesses all these features except the extraction ability. PBS and SWAG have extraction ability but are not capable of hiding environment calls automatically and they do not have an easy implementation of architecture conformance analysis.
Both PBS and Bauhaus's structures and utilities are useful for maintenance and re-engineering of software systems; while Dali ane SWAG are mainly designed for reverse-engineering purposes.

Assessment Criteria	Dali	PBS	SWAG	Bauhaus
Extraction capability				
Integrated extractor	-	++	++	++
Hiding system library calls	+	0	0	++
Parsing ability	0	++	+	+
Ease-of-use	+	+	+	++
Combined views	+	-	-	+
Abstraction capability				
Files-directory decomposition	++	0	++	++
Logical sub-module decomposition	++	+	++	++
Ease-of-use	+	0	+	+
Visualization				
Node types	+	0	++	++
Edge types	+	0	+	++
Combined views	+	-	-	++
Annotations	++	0	++	++
Layouts	+	0	+	++
Navigation	++	0	++	++
Node attributes editable	+	0	+	+
View editable	++	++	-	++
User Interface	+	0	++	++
Views storable	++	-	++	++
History of browsed locations	-	-	0	0
Language support	++	0	+	++
Completeness	+	0	+	+

Table 3: Assessment of the software architecture reconstruction toolkits

++: excellent, +: good, 0: minimal and -: not at all.

6. Conclusions and Future Work

The reconstruction tools evaluated are all quite different with varying strengths and weaknesses. The capability of each software architecture reconstruction tool is evaluated by applying to a C-written ‘concepts’ program. However, no tool is fully automatic since it requires human interaction at some stage to obtain application-specific knowledge about the system.

Dali provides high flexibility in supported language, abstraction ability and integration of new tools into the workbench. It is also good at visualizing architectural information into various layouts. Dali does not have any history mechanism. It supports multi language information.

PBS is the model of building software bookshelves which store system documents on a web server to be shared by many users at the same time. It has its own extractor and the abstraction of data requires a special file specifying the containment of the system’s components. It does not support views and navigation efficiently. PBS does not support multi language information.

SWAG has a nice interface and provides good visualisation capabilities. It also has its own extractor and abstraction ability similar to PBS. However, both PBS and SWAG do not offer functionality to perform the architecture conformance analysis.
Bauhaus toolkit features a lot of new technologies such as layered views, Shrimp views and various layout algorithms. It provides a number of other functionalities for software maintenance purpose apart from full functionalities in software architecture recovery. Moreover, Bauhaus can parse a variety of languages through its analyzers for C, C++, Java and COBOL. Bauhaus is better at visualization while Dali is more extensible.
We assume that the application of architecture reconstruction will be used in a much broader technical sense. This could be used by an organization that is looking to modernize their system with high business value. From source code to architectural views depicts all the unfolded information. This is very useful for white box modernization approach. This could help in constructing architectural view and identify problematic components which need modification.
Although Dali, PBS, SWAG and Bauhaus are prototype tools, they all provide concrete approaches for extraction, abstraction and visualization. However, none of them is complete. Their development should be continued to extend their scope, improve their extraction capabilities and user interaction. Especially, history mechanism should be supported well in all tools to provide users more flexibility in navigating different abstract levels of architecture. An option of playback or 'undo' should be developed to provide the user more ease in browsing the hierarchical presentation of architecture. In addition, analyzing each other's strengths and shortcomings thoroughly as well as integrating new methodologies and approaches towards the architecture reconstruction problem would be helpful.

7. REFERENCES

[1] Amstrong, M. N., and Trudeau, C., "Evaluating Architectural Extractors", *Working Conference on Reverse Engineering, Honolulu, Hawaii*, pp 30-39, October12-14, 1998.

[2] Bauhaus group, "Tour de Bauhaus", http://www.*Bauhaus*-stuttgart.de/demo/index.html, version 4.7.2, December 2003.

[3] Bellay, B., and Gall, H "A Comparison of Four Reverse Engineering Tools", *Proceedings of the 4th Working Conference on Reverse Engineering*, pp 2-11, October 1997.

[4.] Biggerstaff, T.J., Mitbander, B.G., and Webster, D.E., "Program understanding and the concept assignment problem", *Communications of the ACM*, pp72-82 Vol 37, Issue 5, May 1994.

[5] Clements, P., Kazman, R., and Klein, M., Evaluating Software Architecture – Methods and Case Studies. Software Engineering Institute, Carnegie Mellon University, 2002.

[6] Holt, R., "PBS: The portable Bookshelf- Introduction", http://*SWAG*.uwaterloo.ca/*PBS*/.

[7] Holt, R., "Software Bookshelf: Overview and Construction", http://*SWAG*.uwaterloo.ca/*PBS*/.

[8] Imagix Corporation, Imagix 4D, http://www.imagix.com.

[9] Kazman, R., and, Carriere, J., "Playing Detective: Reconstructing Software Architecture from Available Evidence", *Journal of Automated Software Engineering*, pp107-138, April 1999.

[10] Koschke, R., "Readme.txt", *Bauhaus* toolkit distribution, December 2003.

[11]Koschke, R., Eisenbarth, T., and Simon, D., "Locating Features in Source Code", *IEEE Transactions on Software Engineering*, pages 210-224, Vol. 29, No. 3, March 2003.

[12] Murphy, G., Notkin, D., Griswold, W.G. and Lan, E.S., "An Empirical Study of Static Call Graph Extractors", *ACM transaction on Software Engineering and Methodology*, Vol 7, Issue 2, pp 158-191, April 1998.

[13] O'Brien, L., "*Dali*: A Software Architecture Reconstruction Workbench", Software Engineering Institute, Carnegie Mellon University, May 2001.

[14] O'Brien, L., "Experiences in Architecture Reconstruction at Nokia", Software Engineering Institute, CMU/SEI-2002-TN-004, August 2003.

[15] Rigi, http://www.rigi.csc.uvic.ca/index.html, 2000.

[16] Scientific Toolworks Inc., http://www.scitools.com.

[17] SWAG group, "Introduction to SWAGKit",
http://www.*SWAG*.uwaterloo.ca/*SWAGKIT*/#introduction

[18] Taylor R.N., Medvidovic, N., Anderson, K.M., Whitehead Jr., E. J., Robbins, J. E., Nies, K.A., Oreizy, P., and Dubrow, D.L., "A component-and Message-Based Architectural Style for GUI Software", *IEEE Transaction on Software Engineering*, pp390-406, Vol: 22, Issue: 6, June 1996.

[19] Trevors, A., "The Software Architecture Toolkit", University of Waterloo,
http://*SWAG*.uwaterloo.ca/*SWAGKIT*/, February 2003.

[20] Wind River, Sniff+, http://www.windriver.com/products/sniff_plus/index.html

[21] Bass, L., Clement, P., and Kazman, R., *Software Architecture in Practice*, Addison Wesley, 2003.

[22] Holt, R., Schurr, A.,Sim, S.E., and Winter, A., "Graphical eXchange Language",
http://www.gupro.de/GXL

[23] Koschke R., and Simon D., "Hierarchical Reflexion Models," *10th Working Conference on Reverse Engineering*, pp36-46, Victoria, B.C., Canada, November13-17, 2003.

[24] Knodel J., "Process Models for the Reconstruction of Software Architecture Views", University of Stuttgart, http://elib.uni-stuttgart.de/opus/volltexte/2002/1176/pdf/DIP-1987.pdf. July 2002.

[25] Murphy G., and Notkin D., "Reengineering with Reflexion Models: A case study", *IEEE Computer Society Press*, pp 29-36, Vol 30, Issue 8 , August 1997.

[26] Kazman R., O'Brien L., and Verhoef C., "Architecture Reconstruction Guidelines", 3rd Edition, CMU/SEI-2002-TR-034, the Software Engineering Institute,
http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr034.pdf, November 2003.

[27] Tilley S., and Smith D.B , Perspective on Legacy System Reengineering, Software Engineering Institute, Carnegie Mellon University, 1995, available at http://www.sei.cmu.edu/reengineering/lsyree.pdf

[28] Seacord, R. C., Plakosh, D., and Lewis G. A., Modernizing Legacy Systems, SEI Series in Software Engineering, Addison Wesley, 2003.