International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:10, 2008

Recovering Artifacts from Legacy Systems using Pattern Matching

Ghulam Rasool, and Ilka Philippow

***Abstract*—**Modernizing legacy applications is the key issue facing IT managers today because there's enormous pressure on organizations to change the way they run their business to meet the new requirements. The importance of software maintenance and reengineering is forever increasing. Understanding the architecture of existing legacy applications is the most critical issue for maintenance and reengineering. The artifacts recovery can be facilitated with different recovery approaches, methods and tools. The existing methods provide static and dynamic set of techniques for extracting architectural information, but are not suitable for all users in different domains. This paper presents a simple and lightweight pattern extraction technique to extract different artifacts from legacy systems using regular expression pattern specifications with multiple language support. We used our custom-built tool DRT to recover artifacts from existing system at different levels of abstractions. In order to evaluate our approach a case study is conducted.

***Keywords*—**Artifacts recovery, Pattern matching, Reverse engineering, Program understanding, Regular expressions, Source code analysis.

I. Introduction

RECOVERING design information from software is an active research area in reverse engineering. Reverse engineering is defined "as a process of analyzing the program in an effort to create a representation of a program at higher level of abstraction then source code. Reverse engineering is a process of design recovery". [1]. The most important aspect of a successful reverse engineering in aiding users is to understand the domain, functional, structural and implementation of a software system in a particular domain. The methods and tools available to extract the different artifacts at different level of abstractions will not be suitable and sufficient for all users in different domains. The users should be able to choose techniques and way to recover the design artifacts according to specific maintenance tasks at hand at different level of abstractions and integrate other tools and applications that provide complementary functionality, and allow developing their own abstract mechanisms for activities if they require. The primary objective of reverse engineering is to increase the comprehensibility of the system both for maintenance and reengineering activities.

Ghulam Rasool is a PhD student in TU Ilmenau Germany from April 2008. (corresponding author, phone: 0049-017664822342).

Ilka Philippow is working as head in process informatics Group in TU Ilmeanu (e-mail:ilka.philippow@tu-ilmeanu.de).

The software reverse engineering has many contributions to program comprehension, reengineering, maintenance and reusability of existing legacy systems. Understanding software architecture is important for reuse, maintenance and evolution of existing software.

Architecture recovery refers to extraction of information which constitutes architecture elements, styles and patterns. The architecture recovery of large and complex legacy systems is a time consuming activity because of very poor, outdated and inconsistent documentation. Changes are often in the nature of software and have significant impact on its architecture. Due to turnover of developers, lot of knowledge about the system domain is lost. Sometimes the changes that are made by the developers in source code are not updated in the documentations which results inconsistency between source code and documentation. So the only reliable source of information for the developer is the source code. The use of different tools is helpful for extracting useful information from the source code which is further used to analyse the architecture of existing system. The recovery of different architecture and design artefacts is not trivial and the manual search for recovery consumes valuable resources. The existing tools and techniques are valuable but have limitations discussed in next section.

The lexical and syntactic tools are used for extracting different artefacts from source code with their strengths and limitations. The tools are compared on the basis of their pattern matching power, programming power, extraction speed, accuracy and robustness. The major problems with the tools are their limited language support and information retrieval capabilities. The legacy systems that were developed in languages like COBOL, FORTRAN and PASCAL etc, have no support from majority of tools. Syntactic tools are more precise and put no burden on the developer but they do not support the systems with missing header files, having syntax errors and incomplete codes. . Mostly the UML tools are able to create the class diagrams only for object oriented code but these tools are not able to create different type of associations and even miss sometimes the dependency of classes [17].

The lexical based tools are still the best choice for the software engineers to understand the software architecture, and to extract the abstract and required patterns with accuracy and ease. So we used a lexical based patter matching tool DRT [2] using regular expression pattern extraction techniques to extract different artefacts from the legacy systems. One major problem with the lexical tools is their limited vocabulary and language support. We used the

abstraction methodology to design the innovative vocabulary of our tool DRT which is generic in the nature that similar specifications are used to match desired patterns of source code written in different programming languages. Although, there are also certain limitations in our regular expression pattern matching techniques as mentioned by Paul et. al in [3].

II. Artifacts Extraction Methodology

The parse based and regular expression based tools are used to extract different artifacts from the source code and documents and represent it at higher level of abstractions. We preferred the regular expression based extraction due to their simplicity, ease of use, matching power and robustness features. The regular expression extraction technique uses the pattern specifications to extract the desired system artifacts. The hierarchical, nested and abstract specifications are designed to match the required patterns from source code and documents. The regular expression technique is flexible in the sense that it can be applied to different kind of system artifacts including source code (languages) and data files and only syntactic knowledge of the subject is required. The engineer designs the regular expression pattern, match the pattern with the source code and as a result get valuable information which is further used for extracting other patterns. For example the user writes the following regular expression pattern to extract Java methods from the source code:

(JMethodAccessSpecifier)?\s*(JMethodModifiers)?\s*(JPrim Types)?\s*(\w+)\s*(\(\s*(aa)?\)\s*((throws)\s*(\w+))?\s*\{)

The above pattern specification is written according to Java method definition in which JMethodAccess specifier determine whether the method is fiendly|public|protected|Private|private|protected. The JMethodModifiers matches the above pattern with synchronized|native|final|abstract|static which are possible Modifers in Java method definition. JPrimTypes determines the possible return type of method followed by method name, argument types and exception handling.

Similarly following regular expression pattern specification is used to extract comments from C/C++ code files. The documented comments in source code are very valuable source of information especially if documentation and domain information is not available.

(//*(.*))|(\/*(.*)\s(.*))

The sample regular expression pattern specifications for C/C++, Java, Visual Basic, Cobol, and Pascal are presented in [22]. The regular expression patterns are very simple in syntax that user can modify the pattern definition according to the requirements. The architecture of our pattern extraction methodology is shown in Fig. 1.

The user writes the pattern specification using the available documents, domain knowledge and system artefacts. The pattern definition is matched with the source code and we get different source code views and artefacts as a result. The user can again use these views for defining new pattern definitions to extract more artefacts. The pattern view analyser presents the recovered source code model, architectural views, architectural artefacts and different metrics. The file, class, function and variable level of metrics are extracted to analyse the quality of code and for test case generation. Our pattern specifications can be used by the other lexical tools for pattern matching and extracting different artefacts from the source code and text documents.

III. Case Study

Allegiance is a multiplayer online game shipped by Microsoft in 2000. Microsoft has released the source code of Allegiance to the public for research and education purpose. The source code utilizes 512 MB of memory space and has multiple type of files. The source code was easily available to us and has different subsystems and entities. It gave us oppourunity to explore the architectural artifacts from Allegiance game.

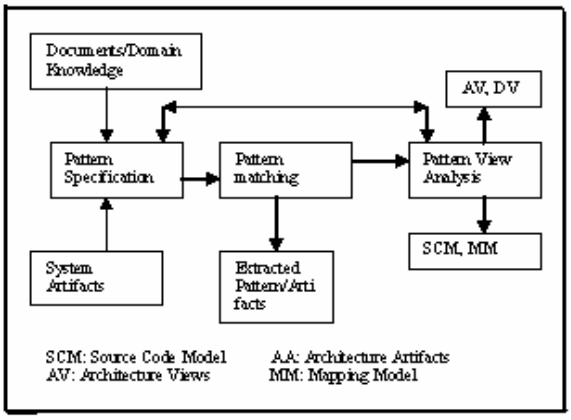

Fig. 1 Pattern Extraction Technique

1) The documentation related information is not available on the web. So after reviewing the software of Game, we extracted its source code information of related C, C++, Java, header and other files for further extracting different type of artifacts as shown in Table I.

TABLE I
Break Down of Files in Allegiance

Folders	C++ Files	Cfiles	Header Files	Java Files	Others
120	589	46	711	0	6283

2) Secondly, we identified the major folders and entities in order to associate them with each other. The important folders identified were Utility, AGC, Allguard, AllsrvUI, Club, Edit, MSRGuard and some others. We mapped the entities to the source code to associate the entities and sub-entities with them in order to develop the high level model iteratively. High level model gave us overview about the structure of the system and help in building hypothesis for further pattern specifications.

3) Later, we extracted the comments from the source code because documentation is not available. These comments gave us further clue to explore the different entities and functionality of the game. The comments extracted are shown in Fig. 2.

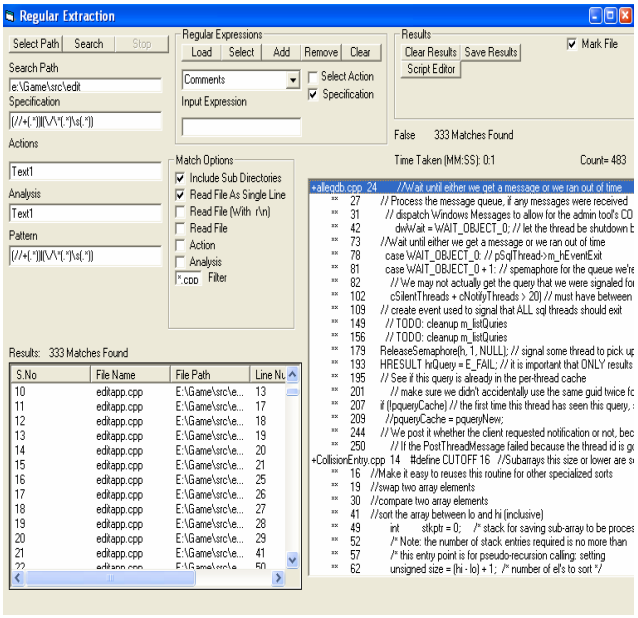

Fig. 2 Extracted Comments

4) Next the mapping was performed using regular expressions. The mapping associated the entities with the available source code through mapping iteratively by different pattern specifications. Fig. 3 shows the results of different mapped classes and functions.

5) Next we can recover the class diagram and use case diagrams from the Allegiance Game AGC entity with the help of DRT and other reverse engineering tools like Rational Rose. The recovered class diagram is not shown here in this paper and can be seen in [23].

6) Finally, the other useful artifacts from AGC entity are extracted which can be used for further pattern definitions and recovery of other useful artifacts of the Allegiance Game as shown in Table II.

TABLE II
EXTRACTED ARTIFACTS WITH TIME SPENT ON EXTRACTION

Artifacts	C/C++ Files	Header files	Time Taken mm:ss
Classes	99	67	0:8
functions	1299	1846	1:16
variables	3109	1818	0:42
statements	33554	6193	2:09
comments	39759	26098	1:0
structures	90	550	0:10
define	330	4315	0:18
Loop st	1230	197	0:13
enum	17	50	0:9

In another study, we extracted different patterns from Talk2Me which is implemented in Java. The Talk2Me is an instant messenger developed by Svetoslav Chekanov [24], a software engineer of India having characteristics like yahoo and other messengers. Its source code is available free of cost for the public. We used it for recovering different types of artifacts as shown in Table III with the Pattern definitions.

TABLE III
ARTIFACTS FROM TALK2MESSANGER WITH PATTERNS

Artifacts	Pattern Specification	Matched results
LOSC	^(.*)	4972
Comments	(/*[\d\D]*?*/)|(//+)	160
Classes	(JClasModifiers)?\s*((Class)((extends)\s*(\w)+)?\s*((implements)\s*(\w)+)?(\s*(,)\s*(\w+))*\s*\{)	23
methods	(JMethodAccessSpecifier)?\s*(JMethodModifiers)?\s*(JPrimTypes)?\s*(\w+)\s*(\(\s*(aa)?\)\s*((throws)\s*(\w+))?\s*\{)	156
Jinterface	((public)|(abstract))?\s*interface\s*(\w+)\s*(extends)\s*(\w+)\s*\{	0
Blank Line	^[\s]*$	561
Assign st	(JPrimTypes)?\s*(VrNam)\s*(COP)?(=)\s*((VrNam)|(\d+))|(Expression)/;	340
Variables	(CTypes)\s*(VrNam)(\s*(=)\s*((\d+)|(\w+)))?((\,)(VrNam)(\s*(=)\s*((\d+)|(\w+)))?)*\;)	116

Our regular expression pattern specifications are generic and tested on source code of different programming languages to extract different artifacts which are helpful in extracting other artifacts.

IV. RELATED WORK

Reverse engineering of legacy systems is time consuming and challenging task which can be assisted by different tools. Without tool support it is very difficult to understand the structure of the large and complex systems. Different researches have used different lexical, syntactic and semantic analysis tools to support their work in reverse engineering. The most well known lexical tools like GREP, AWK, MAWK [4], and LSME [5] are available with their best features and certain limitations. The syntactic tools like REFINE [6], SCRUPLE [7], A*[8], GENOA [9], TAWK [4] are used to analyze the different software systems. The active research groups are developing new tools and extending functionality of their existing tools. Information about number of tools used by engineers is available on Web in [10]. The major problem with these tools is their limited language support and vocabulary.

The selection of the tools depends upon the requirements of the users and characteristics of the tool. The commercial tools are very expensive and open source are not best suited to the requirements of user. The usability of open source tools is also an issue. So this gave us motivation for developing and using our own custom built tool DRT. The Imagix 4D [11], Rigi

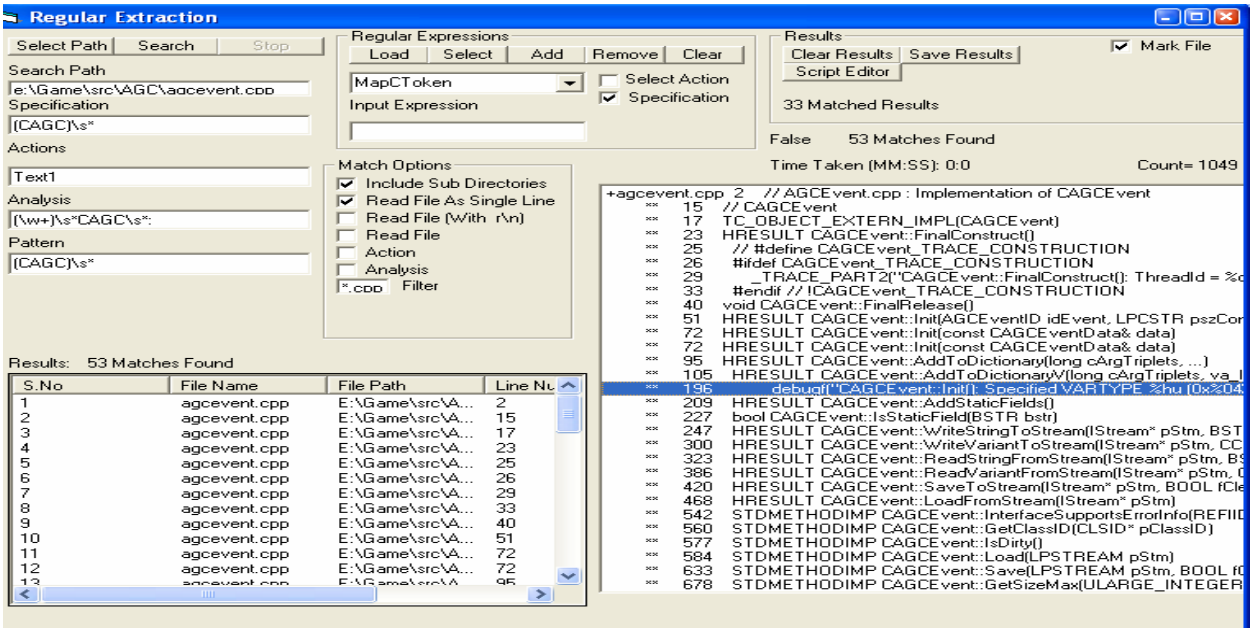

Fig. 1 Mapped entities

[12], Alborz.[13], PBS[14],Bauhaus[15], Dali Workbench [16] are similar to our tool DRT but differ in numerous features as mentioned in [25].

In architecture recovery much work has been on techniques which combine top-down and bottom-up approaches. In bottom-up reverse engineering tools are used to extract source models and in top-down queries are applied to extract expected patterns [17]. Harris et al. [18] outlined a framework that integrates reverse engineering technology and architecture style representation. Guo et al. [19] used an iterative and semi-automated architecture recovery method called ARM. Startipi et al. [20] outlined Alborz which use source model and queries as basic inputs for architecture recovery. Pinzger et al. [17] used simple string pattern matching techniques for extracting different artefacts but without support of action and analysis pattern definitions. Phillipow et al. [21] used design patterns for extracting information from source code by using different tools.

The [17, 18, 19, 20, 21] approaches are similar to our approach that all use patterns for architecture recovery. Our approach is different from these approaches in pattern definitions, abstraction and extraction. The above mentioned techniques use reverse engineering tools which extract patterns containing architecture elements. These reverse engineering tools are not easily available, have many limitations and are sometimes time consuming for first stage of architecture recovery. So we used our own custom build simple tool DRT [2] using abstract regular expression pattern definitions. We start pattern definitions at low level and refine the patterns iteratively to extract the desired architecture and design information. Our tool will be integrated with the other tools which are under development to represent the extracted artifacts at different levels of abstractions.

V. Conclusion and Future Work

The pattern matching and processing tools help the programmer to understand the existing code and make appropriate changes. The selection of pattern matching tools depends on the key properties of expressive power, flexibility, scalability, performance, versatility, speed, accuracy and robustness etc. It is very difficult to achieve the effective balance between all the above mentioned properties in any tool. We used the lexical based pattern matching technique using Extended Regular Expressions to extract the desired artefacts from source code of different languages with speed and accuracy. We designed the creative and innovative vocabulary of our tool to obtain more precision in our pattern matching technique. Our technique has successfully recovered the architectural artifacts from the legacy system architecture. In future, research is focussed to the following areas.

- Extension of vocabulary of our tool to support larger and mix coded systems.
- Recovery of dynamic view of the system.
- Specification of our pattern language so that it can handle all type of design patterns (Creational, Structural and behavioural).
- Integration of automata theory and formal methods for artifacts recovery.

References

[1] Pressman, Roger S., Software Engineering: A Practitioner's Approach, McGraw Hill, 1997.
[2] Ghulam Rasool, Nadim Asif, " A Design Recovery Tool", International Journal of Software Engineering, ISSN(1815-4875), Vol 1, pages 67-72, May 2007
[3] Paul, S. and Prakash, "A framework for Source Code Search using Program Patterns". IEEE Transactions on Software Engineering, vol 20, pp 463-475, 1994.
[4] Tony Abou-Assalaeh and Wei Ai, "Survey of Global,Regular Expression Print(GREP) Tools", March 02, 2004.

[5] Darren C. Atkinson, William G. Griswold, "Effective pattern matching of source code using abstract syntax patterns", SP&E, pp 413-447, December, 2005 (www.interscience.wiley.com). DOI: 10.1002/spe.704.
[6] Burson S, Kotik GB, Markosian LZ, "A program transformation approach to automating software re-engineering", IEEE Computer Society Press: pp 314–322, Los Alamitos, CA, 1990.
[7] Paul S, Prakash A. "A Query Algebra for Program Databases". IEEE Transactions on Software Engineering 1996; vol 2 pp 202-217.
[8] Ladd DA, Ramming JC. " A* A language for implementing language processors" IEEE Proc on Software Engineering, vol 21, pp894–901,1995
[9] Devanbu P.T. GENOA—"A customizable, language- and front-end independent code analyzer", Procs of the 14th International Conference on Software Engineering", Melbourne, Australia, ACM Press New York, pp 307–317, 1992.
[10] [http://kilana.unibe.ch:8080/SCG/370 [Accessed on 10th June 2007].
[11] Imagix4D,UR=http://www.imagix.comImagix Corp. [Accessed 10th march, 2007].
[12] Rigi,URL=http://www.rigi.csc.univ.ca/rigi/rigiindex.html [Accessed 5th march, 2007].
[13] Kamran Sartipi, Lingdong Ye and Hossein Safyallah, An Interactive Toolkit to Extract Static and Dynamic Views of a Software System, Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)0-7695-2601-2/06.
[14] P.J. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, et al. The software bookshelf. IBM Systems Journal, 36(4):564–593, November 1997.
[15] Meena Jha1, Piyush Maheshwari, Thi Khoi Anh Phan, A technical Report On, "A Comparison of Four Software Architecture Reconstruction Toolkits", UNSW-CSE-TR- 0435, October 2004.
[16] Bauhaus group, "Tour de Bauhaus", http://www.Bauhaus-stuttgart.de/demo/index.html, version 4.7.2, December 2003.
[17] Martin Pinzger, Harald Gall, Pattern supported architecture recovery, In proceeding of 10th International Workshop on program comprehension (IWPC'02).
[18] D. R. Harris, H. B. Reubenstein, and A. S. Yeh. Reverse engineering to the architectural level. In Proc. of the 17thInternational Conference on Software Engineering, pages 186–195, Seattle, Washington, April 1995. ACM Press.
[19] G. Guo, J. Atlee, and R. Kazman. A software architecture reconstruction method. In Proc. of the 1st Working IFIPConference on Software Architecture, pages 225–243, San Antonio, Texas, February 1999. Kluwer Academic Publishers.
[20] K. Sartipi, K. Kontogiannis, and F. Mavaddat. A pattern matching framework for software architecture recovery andrestructuring. In Proc. of the 8th International Workshop on Program Comprehension, pages 37–47, Limerick, Ireland,June 2000.
[21] Ilka Philippow, Detlef Streitferdt, Matthias Riebisch, Sebastian Naumann, An approach for reverse engineering of design patterns, Accepted: 29 January 2004/Published online: 29 April 2004 –. Springer-Verlag 2004.
[22] Ghulam Rasool, Nadim Asif, "Software Artifacts Recovery Using Abstract Regular Expressions", In Proc of 11th IEEE Multitopic Conference, 28-30 December 2007 , Comsats Institute of IT Lahore Campus.
[23] Nadim Asif, "Software Revere Engineering", ISBN 969-9062-00-2 Pakistan.
[24] http://www.codeproject.com/useritems/talk2me.asp [Accessed on 22th April 2008].
[25] G. Rasool, N. Asif" International Journal of Software Engineering of software Engineering", Vol 1, No.1. pp. 67-71, May 2007.

Ghulam Rasool is a PhD student at TU Ilmenau Germany. He did his Masters in Computer Science from Bahauddin Zakariya University, (A public sector University) in Multan, Pakistan. Mr Rasool is doing extensive research in the area of software reverse engineering, architecture recovery and reengineering.

Ilka Philippow is a full Professor for Process Informatics at the Ilmenau Technical University since 1992. In the eighties she was working in the field of software development for technical and embedded systems. She received the PhD in Computer Science in 1981 and finished her habilitation in 1989 at the Technical University of Ilmenau, Germany. She is doing research in the area of pattern recognition, architecture recovery and traceability link management.