Artifacts Recovery and Understanding Using High Level Models

NADIM ASIF, FAISAL SHAHZAD, NAJIA SAHER, RAFAQUET KAZAMI, WASEEM NAZAR
The Islamia University of Bahawalpur
Dept. of Computer Science
Bahawalpur
PAKISTAN
nasif@softresearch.org, faisalsd@gmail.com, najia_saher@hotmail.com,rafaquetkazmi@gmail.com, waseem@uol.edu.pk

Abstract:- The systems are required to understand and present at higher levels of abstractions to perform changes or re-engineer to meet the current requirements. The software systems drift away from the existing implemented source code and the documentations due to the changes. The high level models are used for the purpose of recovering the artifacts and understanding the system to perform the maintenance activities. This paper presents an approach to develop the high level models from the existing source code and documents.

Key Words:- Design Recovery, Re-Engineering, Reverse Engineering, , Program Understanding, Software Maintenance

1. Introduction

The software systems are evolved and changes are performed in the systems to meet the current and future requirements of the users. The software engineers perform different maintenance activities by extracting the different types of artifacts at different levels of details. The artifacts exist at implementation, structural, functional and domain abstraction levels. The changes are performed in the software systems and the existing documents are drifted away form the implementation and fail to represent the current implementation of the system. The reverse engineering techniques help to represent the software systems at higher levels of abstraction than code to recover the desired artifacts, understand and comprehend the source code and elaborate the functionality of the software systems to plan, design and execute the different types of maintenance activities. The software engineers also draw the sketches and diagrams in different formats to represent the systems at higher levels of abstraction for understanding and representing software systems for maintenance activities. The high level models represent the higher level of abstraction of a system in a particular domain. These high level models provide a hint to explore, search, understand the code, functionality and behavior of software systems for maintenance tasks at hand [1,2,3,4,5,6].

2. Background

An entity define/comprehend a concept and is used to represent higher abstraction level of components or modules, data sources and processes in a domain, which are used in the high level models to represent the software systems. The directed arcs are used to represent the relationships and flow of information among the components/modules, data sources and processes. The sub-entities represent the lower levels of abstractions as compared to an entity. For example account is an entity in a banking domain and personal account, corporate account are examples of sub-entities represent the specific types of accounts.

For example, a high level model formed for the Unravel system (used to measure the quality of the code) is presented in Figure 1. The high level model of Unravel developed iteratively in the study and identified three main component of a system. The system performs functions with the help of these three main components: a source code analysis component, a link component, and an interactive slicing component.

The analysis component collects from source files (with a .c extension) and included header files (usually with a .h extension) the information necessary for the computation of program slices. The information is translated into a representation independent source language called language independent format (LIF). The analyzer is designed like a compiler with a scanner to break the source code into tokens that are recognized by a parser, but instead of generating object code, it produces LIF code. The analyzer also produces a tally of objects (**.**T file) such as procedures and variables, and a file to

list global objects (**.**H file) declared in each included header file.

The link component operates in two parts. The first part, map identifies for each program in the current directory its constituent files and then saves this information in a file named SYSTEM. The second part of the link component, slink, uses the SYSTEM file to merge data-flow information from the **.**LIF, **.**T and **.**H files created from separate source files into a single **.**LINK file and a single **.**K file. Under user control, the interactive component extracts and displays program slices and keeps a record of user activities in a .LOG file.

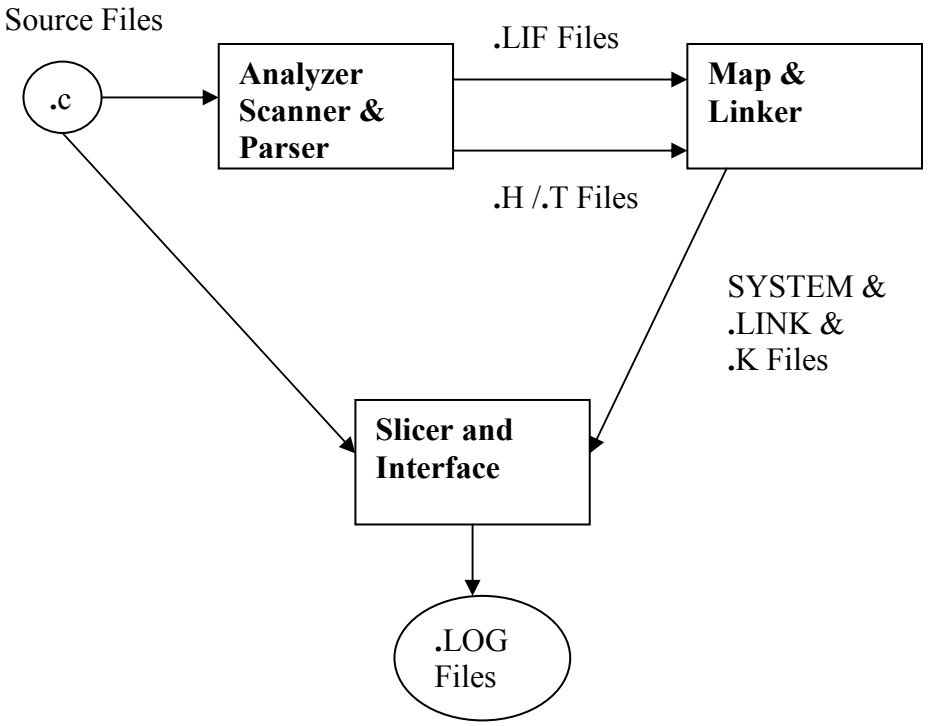

Figure 1 High Level Model of Unravel

2. Identification of Entities

The high-level models are developed using the domain knowledge, personal experience, application users, available maintenance personnel's, existing source code and available documents (specifications, designs, manuals). The software engineers in the first step identify the entities using the available information and then associate them through arcs and label the arcs to mark the flow of specific information from one entity to another entity. For example the engineer initially identify the entities ***parser***, ***token***, ***tag*** and ***scanner*** to develop the high level model of Mozilla HTML parser through his experience and knowledge about the domain. The software engineer maps these entities to the source code and the documents to associate the entities and sub-entities with them to develop the high level models iteratively.

3. Existing Source Code

The available source code exist in many forms; may be written in multi-languages or have different dialects and scripts, can not be compiled or have errors and complete code is not available. The software engineers debug the source code and find the relationships and functionality and associate them with relevant entities to develop the high level models. It is a time consuming and laborious task and even in the case of large systems become very difficult and expensive. The developers and the software maintenance personnel record the functionality and changes performed in the system in the form of comments in the source code. The engineer in the second step extracts the history facts (comments - which are buried in the source code) from the available source code which represents the system truly. The history facts help to identify the main and sub-entities of the system, and the functionality of the system it performs. The figure 2 depicts the REAM tool used to extract the history facts from the HTML parser source code.

4. Available Documents

The available documents exist in many formats and have specific objectives to represent the systems. These documents (e.g. specifications, design documents and manuals) are also drifted away from the existing implementation (than actual available source code) and do not represent the system. The entities are also mapped to the documents (if electronically available) to identify more descriptions about their functionalities in the system. This step also helps to build the knowledge about the entities of the system in more details and their relationships among them.

5. Mapping

The mapping step associates the entities with the available source code and documents through mapping iteratively. The mapping is performed using the regular expressions. It allows the engineer to define the mapping patterns of its own choice required by the tasks to map to the multi-language, different dialects/scripts, having errors or incomplete source code. Initially the identified entities found in the first and second step are mapped to the source code. The identified sub-entities are further associated with the lower level entities through mapping, which constitutes the sub-entities of a particular domain.

Let $\boldsymbol{S}$ represents the available source code, which exist in many forms. The software source code is composed of components $\boldsymbol{C_n}$ (or modules) and components consists of functions which are represented by $\boldsymbol{f_m}$. The $\boldsymbol{D}$ represents the available documents, which provide the specific details about the system.

Let $\boldsymbol{E_n}$ represents a set of entities in a particular domain $E_n = \{ E_1, E_2, E_3 \ldots\ldots E_m)$, which are implemented by using a multi-language, different dialects/script in different periods of time $\boldsymbol{T_n}$ (development and maintenance) to perform certain functions.

The available source code is organized in the form

```
**********************************************************
 This the ITagHandler deque deallocator, needed by the
 CTagHandlerRegister
 ***********************************************************/
 This funtor will be called for each item in the TagHandler que to
 check for a Tag name, and setting the current TagHandler when it is reached

**********************************************************/**
**********************************************************
 This a an object that will keep track of TagHandlers in
 the DTD.  Uses a factory pattern
**********************************************************/***
**********************************************************
 The CTagHandlerRegister for a CNavDTD.
 This is where special taghanders for our tags can be managed and called from
 Note:  This can also be attached to some object so it can be refcounted
 and destroyed if you want this to go away when not imbedded.
**********************************************************
******/CTagHandlerRegister gTagHandlerRegister;
**********************************************************
 And now for the main class -- CNavDTD...
 ***********************************************
 *  This method gets called as part of our COM-like interfaces.
 *  Its purpose is to create an interface to parser object
 *  of some type.
```

Figure 2 Extracted History facts from the HTML parser source code

of different types of directories and files physically. Let A_p is a set of physical association of source code files. The A_c is a set of conceptual association of source code and A is a super set of A_p and A_c. The associations among physical and conceptual associations are represented by conceptual view (V_c), physical view (V_p) and relational view (V_r). The V_s = {V_c, V_p, V_r} represents the set of software views.

Conceptual Views (V_c): Association of entity with different components/modules, classes, routines / functions to represent the certain functionality.

Physical View (V_p): Association of entity with source codes organized in different files and directories. The file name, file types (identified usually through file extensions) and directory name represent the nature of the source code it contains.

Relational View (V_r): Relationships among different artifacts like components, functions and variable.
The figure 3 depict the map of *CToken* and *sPasrer* entities to the HTML parser files. The *CToken* entity relationship with the other artifacts in the source code is shown in figure 4.

Map	To	Files
\sCToken\s	C:\TestedData\Mozilla8\HTMLParser	*.*
\sParser\s	C:\TestedData\Mozilla8\HTMLParser	*.h
.	.	.
.	.	.

Figure 3 Mapping Entities to HTML Parser Code

6. Source Code Model

The source code model is extracted by mapping the entity to the source code, which represent the domain information of this entity implemented in the source code to perform some functions. The source code model also represents the entity associations to the components/modules, sub-components, classes, functions and variables, which represent the low-level implementation details of the source code. In figure 4, the numbers represent the line number of that particular file where the mapped artifact exists. The mapping associates the CToken entity with all the classes and functions of the HTML parser source code. The result of this mapping is a source code model which represents the relationship of CToken entity with other artifacts (Classes and function).

The source code model also associates the entities with the directories (in which relevant codes are organized) and the files.

7. Naive Bayesian Classifier

The Naive Bayesian classifiers are statistical classifiers. They can predict entity relationships probabilities, such as the probability that a given code sample represents the particular entity.
Let X be a code whose entity E, which represent the code is unknown.
Let H be some hypothesis such that X belong to a specific entity E.
For high level model, we determine P(H|X), the probability that the hypothesis H holds given observed code X.

Suppose that X code is a search routine and every time it divides the data into two portions using the pivot for search, and that H is the hypothesis that X is a binary search. Then P(H | X) reflects our confidence that X is a binary search given that we have seen the X code and it search the data. The P(H) is the prior probability. In this example, this is the probability that any given code is a binary search, regardless of how the code (data sample) looks.

The posterior probability, P(H | X) is based on more information (such as background knowledge, available documents, experience) than prior probability, P(H) is independent of code X (data Sample). Similarly, P(X | H) is the posterior probability of code X condition on H. It is probability that X is a search code that we know that it is true X is a binary search code. P(X) is the prior probability of X. Using this example; it is the

probability that a code sample from a set of codes (files) is a binary search code.
The P(X), P(H) and P(H | X) will be estimated from a given code, available documents, experience and knowledge about the application and domain. The Bayes theorem provide a useful way to calculate the posterior probability P(H | X) from P(X | H), P(X) and P(H), Bayes theorem is

$$P(H|X) = \frac{P(X \mid H)\, P(H)}{P(X)} \quad \ldots \text{Eq (1)}$$

Then Bayes Theorem is used in the Bayesian classifier.

1. Code is represented by $X = (x_1, x_2, x_3 \ldots\ldots x_n)$

2. Suppose that there are m entities (E_1, E_2, E_3 ……E_m). Given an unknown code X (i.e. having no entity name). Then classifier will predict that X belong to the entity having posterior probability, condition on X that is, the naïve Bayesian classifier assign an unknown sample code X to the entity E_i if and only if

$$P(E_i \mid X) > P(E_j \mid X) \text{ for } 1 \le j \le m, j \ne i$$

Thus we maximize $P(E_i \mid X)$. The entity E_i for which $P(E_i \mid X)$ is maximize is called the maximum posterior hypothesis . By Bayes theorem Eq(1).

$$P(E_i|X) = \frac{P(X|E_i)P(E_i)}{P(X)} \quad \ldots\ldots\text{Eq (2)}$$

3. As P(X) is constant for all entities, only $P(X \mid E_i)$ $P(E_i)$ need be maximized. If the entities prior probabilities are not known, then it is commonly assumed that the entities are equally likely, that is $P(E_1) = P(E_2) = P(E_3) \ldots\ldots = P(E_m)$, and we would therefore maximize $P(X|E_i)$, otherwise maximize $P(X|E_i)$ $P(E_i)$. The entity prior probabilities may be estimated by $P(E_i) = S_i / S$, where S_i is the number of code sample of entity E_i and S is the total number of code samples.

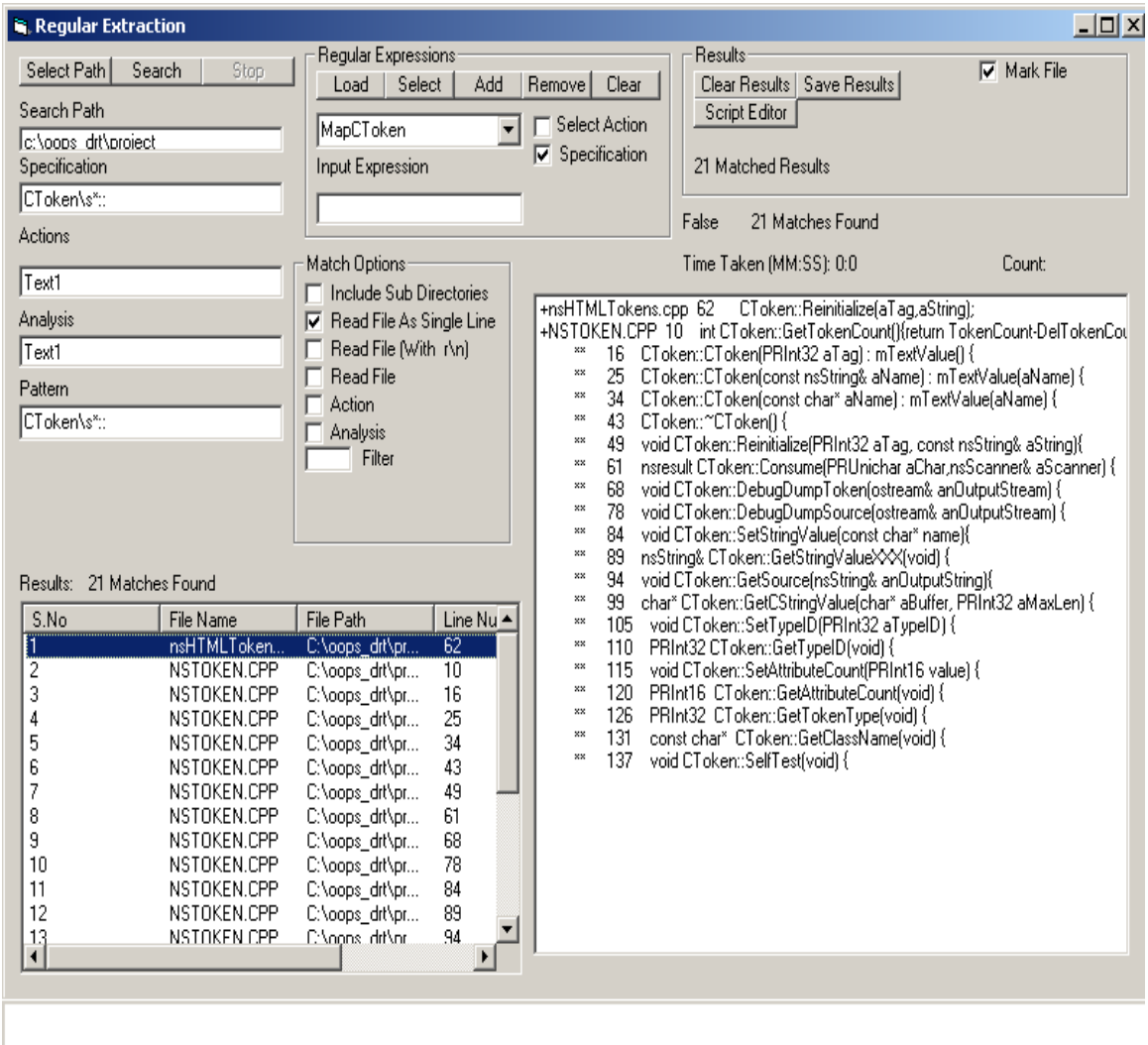

Figure 4 Result of Mapped CToken Entity

4. Given data set with many attributes, it would be extremely computational expensive to compute $P(X \mid E_i)$. In order to reduce computational in evaluating $P(X| E_i)$, the naive assumption of entity conditional independence is made. This presumes that the values of attributes are conditionally independent of one another, given the entity of the sample, that is, there are no dependent relationships among the attributes. Thus,

$$P(X|E_i) = \prod_{K=1}^{n} P(x_k| E_i) \quad \text{..... Eq (3)}$$

The probabilities $P(x_1| E_i)$, $P(x_2| E_i)$, $P(x_3| E_i)$, ... $P(x_n| E_i)$ can be estimated from the code samples, where

a) If A_k is categorical, then $P(x_k| E_i) = S_{ik} / S$ where S_{ik} is the number of code samples of Entity E_i having the value x_k for A_k and s_i is the number of code samples belonging to E_i.

b) If A_k is continuous-valued, then the attribute is typically assumed to have a Gaussian distribution so that

$$P(X|E_i)=g(x_k, \mu_{E_i}, \sigma_{E_i})= \frac{1}{\sqrt{2\pi}\,\sigma_{Ei}} e^{-\frac{(x_k - \sigma_{E_i})^2}{2\sigma^2_{Ei}}} \quad \text{.... Eq(4)}$$

Where $g(x_k, \mu_{E_i}, \sigma_{E_i})$ is the Gaussian (normal) density function for attribute A_k, while μ_{E_i} and σ_{E_i} are the mean and standard deviation, respectively, given the value attributes A_k for code samples of entity E_i.

5. In order to identify an entity for an unknown code sample X, $P(X \mid E_i)\, P(E_i)$ is evaluated for each entity E_i. Code sample X is then assigned to the entity E_i if and only if

$P(X \mid E_i)\ P(E_i) > P(X \mid E_j)\ P(E_j)$ for $1 \le j \le m, j \ne i$

In other words, it is assigned to the entity E_i for which $P(X \mid E_i)\ P(E_i)$ is maximum.

8. DRT for High Level Models

A Design Recovery Tool (DRT) is used to recover the high level models and the tool has the set of attributes: flexibility, extensibility and scalability [7,8,9,10].

Flexibility: The subject system's implementation plays an important role in the recovery of high level models. Many issues exists, which are related to the source code; the language dialect or variant, the robustness of extraction (lexically or parsing) mechanism used (e.g., whether or not it support implementation extractions, syntactically incorrect constructs, or incomplete code fragments), and whether or not mixed-mode source code is supported. For example, C & C++ programs may be written in different dialects, the programs may use the C, C++, Java and different types of scripts to perform the required functions, and COBOL programs may includes database preprocessor directives.

The tools should be flexible enough to support the various activities to recover the high level models tasks. The tasks require different types of system artifacts (i.e. Use Cases, Classes, functions) to abstract at higher levels of abstraction to perform tasks. The artifacts construct and levels at which need to abstract varies from task to task. Most of the tools produce documents/information, which is not relevant to the task. The tools need to support the user specifying the required artifacts, mapping the artifacts, extracting, abstracting and presenting the artifacts in using the available domain information, user experience and knowledge, and adapt the tailored process for available tasks.

Artifacts specifications: The different types of system artifacts are required for different tasks in different domain. The users are required to customize the required system artifacts for the task at different levels to perform the tasks. It is not possible for the tool developers to make it available every kind of artifacts according to the user specific required task. The solution exists in it that the tools should allow the user to specify the artifacts construct for extraction and abstraction of the system artifacts.

Mappings: The user task specific artifacts mapped to the source code or documents for extraction and abstraction purpose. The mapping limits the extraction process scope by improving the extraction performance; specifying the required artifacts constructs.

Extract & Abstract: The recovery of high level models activities requires extraction and abstraction of the system artifacts according to the task at different levels of abstractions. In practice, it is difficult to find at what level the extract and abstract of artifacts need to perform for the available task.

Presentation: The extracted and abstracted artifacts need to be presented in a particular or required format. In the case of large systems, the numbers of artifacts are in big numbers and have different types of relationships, and variety of constructs makes it difficult to present the system artifacts. DRT extract, abstract and present the artifacts in different formats at different levels, which are very much relevant to the task at hand.

Extensibility: The tool for the high level model recovery requires that users can extend the systems functionality by adding different tools and scripts in extraction and abstraction components. The user scripts and code extensions for analysis in the recovery process improve the extraction, abstraction and presentation of system artifacts at different levels of abstractions.

Scalable: The DRT tool can be applied to large systems and different types of source code (languages) and dialects, and provide the extraction, abstraction and presentation of the system artifacts for the required task. For example, not all software artifacts need to be stored and some artifacts may be ignored. Coarse-grained artifacts can be extracted, partial systems can be incrementally investigated, and irrelevant parts can be ignored to obtain manageable artifacts.

In practice, the source code exists in different forms; have different languages code, dialects or variants and different scripts, incomplete, cannot be compiled. The tools are required to provide the support for the extraction and abstraction of source codes at different levels in the recovery process. The tools also support the user in specifying the required artifacts for extraction and abstraction purpose at different levels of abstraction, instead of extracting all the system artifacts for a required task.

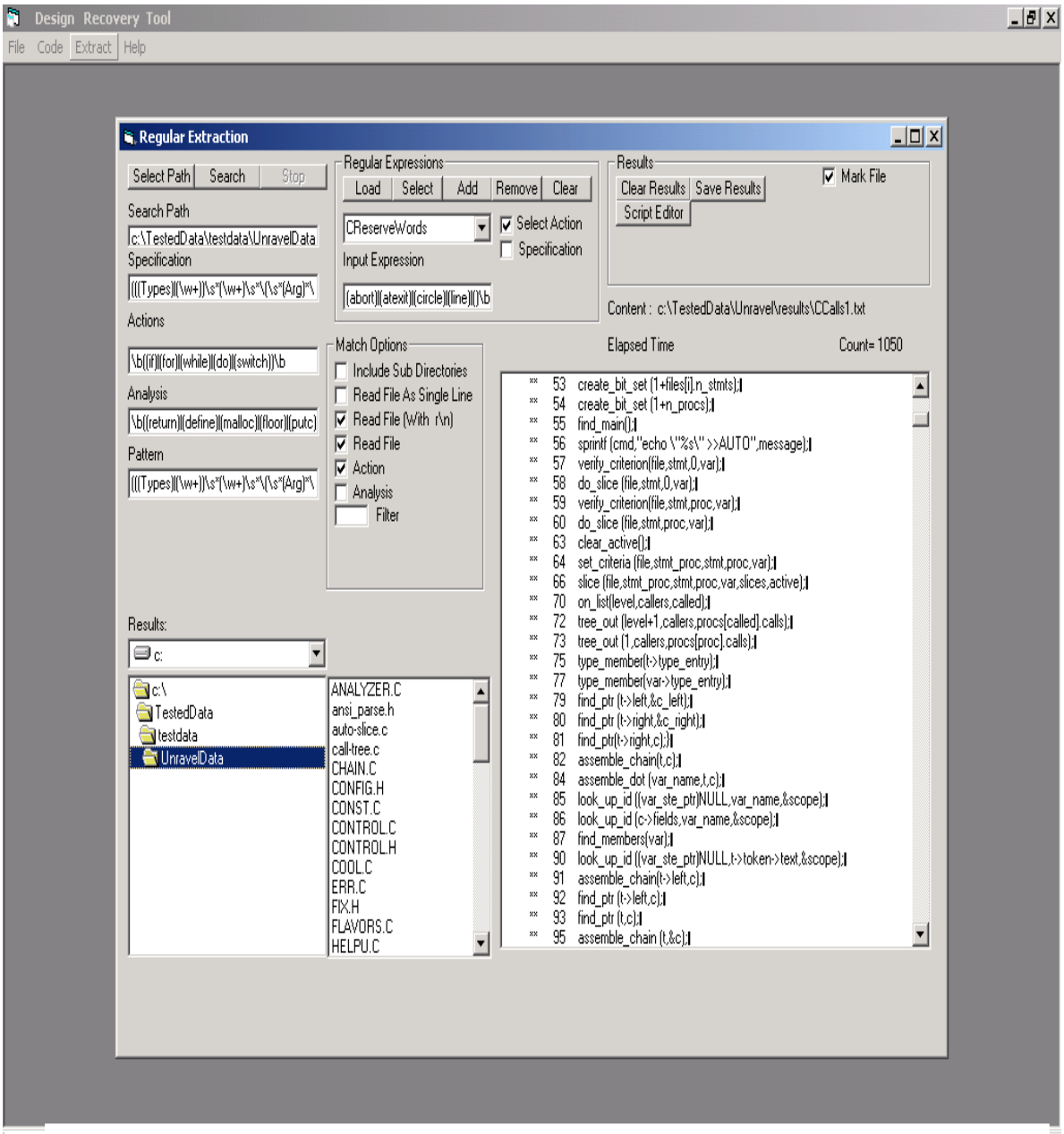

Figure 5 Unravel function calls view using the DRT tool

For extraction and abstraction purpose, tool helps to define the user inputs specification language (pattern, action, and analysis definitions) and mapping entities for extraction and abstraction purpose. For example, initial specification defined by the user was "**<Types> class <ClassName>**" to extract the class names from the source code. The words in angle brackets means the abstracted types, i.e. Types represent the type of the class that can be public, private or protected.

Pattern: The user specifies the information to extract from the system artifacts as patterns. Each pattern uses regular expressions to describe the artifact construct that may be found within the system artifacts. For example, the engineer used the above defined class specification to design the regular expression pattern "(class)\s*\w+\s*" to extract the name of all the classes from the source code as depicted in figure 6.

Action: A user may attach the action to the pattern to be executed when a pattern is matched in the source code. The action code performs operations such as controlling the matching of the constructs in the source code to particular patterns. Specifically, a user may reject matches to a particular pattern by invoking the regular expression within the action. This control is used to reject matches when patterns are too general.

Analysis: In certain cases, the desired artifact cannot be extracted directly during the scanning of the source code. The required artifact can be extracted at the conclusion of extraction from multiple types of information extracted from the system artifacts. A user defines the desired extraction pattern in an analysis section of the specification and further extraction is performed on the intermediate results produced from previous extraction. The tool also processes the abstract information and passes this information as input to the extractor and abstractor components for further processing. It helps to define, store, retrieve and compile the abstract regular expressions and mappings for use in the extraction and abstraction process.

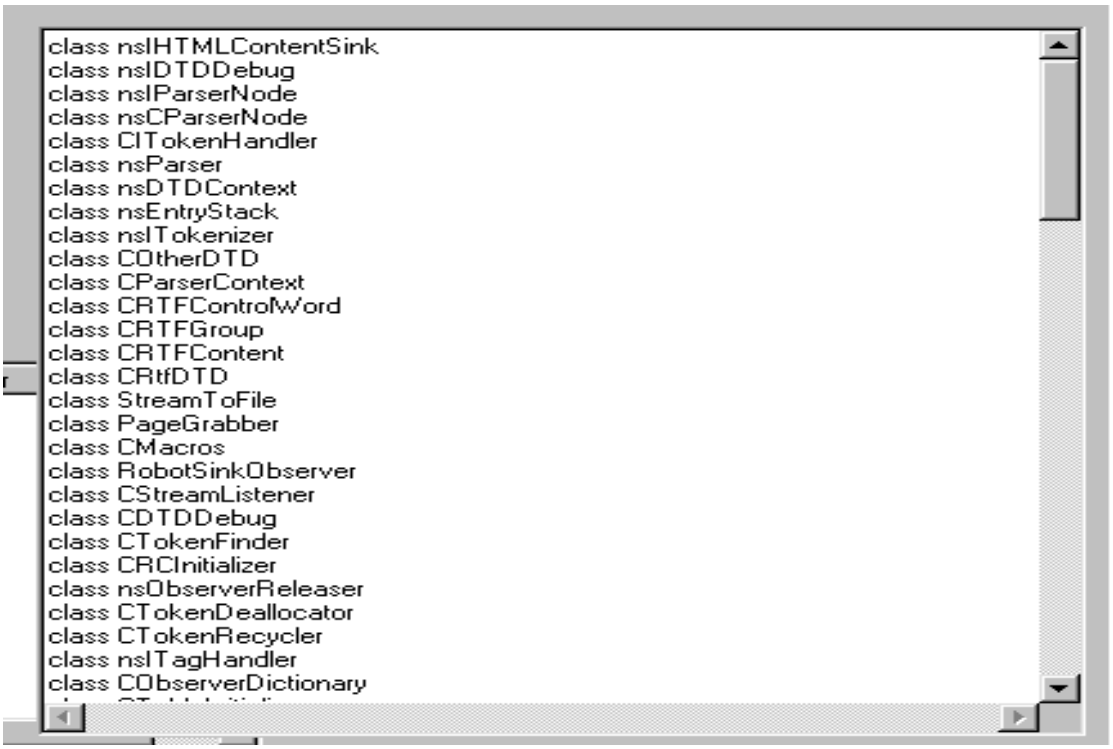

Figure 6. Pattern is applied to extract the Mozilla HTML parser classes

The Abstract Regular Expression Pattern (AREP) represents the regular expressions of high-level concepts or artifacts. The user designs the abstract regular expressions pattern by using the regular expressions and uses the reserve words to name the abstract regular expressions. The abstract regular expression patterns allow the user to define the complex patterns required by the recovery tasks for different languages and dialects using the pattern specification.

Some of the examples of abstract regular expression patterns are presented which are designed to extract the artifacts from source code for recovery of high level models. The regular expressions are used as patterns to design and abstract the complex patterns to represent the different artifacts of interest. More Abstract regular expression patterns can be designed using these patterns. For example, in the given below **Types** abstract regular expression pattern, the word "**Types**" is a name of the pattern and it is separated by special character ' –' from the regular expression, which represent the types used in the source code.

Ansi -s|\w|\d|/|"|!|\(|\)|\\|@|#|\$|%|&|*|\^|:|;|'|\, |\.|\?|\+|-|\=|~|`|\||\[|\]|<|>|_|{|}

Vartypes- char||int|void|float|static|double|long|short|

Types-public|private|protected

Arg-\s|\w|\d |_|,|+|-|/

Args-(Arg)*

Stm-\s+|\w+|\d|/|"|!|\(|\)|\\|@|#|\$|%|&|*|\^|:|_|;|'|\,|\.|\?|\+|-|\=|~|`|\||\[|\]|<|>

Stms-(Stm)*

Structs-(struct\s*\w*\s*\{(Arg)*\s*\})

Enum-(enum\s*\w*\s*\{Stms\}\s*(Arg)*\s*;)

IncludeFiles-#\s*include(.*)[<|"""](.*)[>|"""]

Define-#\s*define\s*(Arg)*

CfunDef-(((\w+)\s+|(\w+))(\((\w+)*\)\s*(Arg)*\{))

CFunCalls-(((Types)|(\w+))\s*(\w+)\s*\(\s*(Arg)* \s*\) \s*;))

Class-((class)\s*(\w)+\s*\{)

IndependentClass-(class)\s*(\w)+\s*\{

Deriveclass-((class)\s*(\w)+\s*:\s*(Arg)*\s*\{)

Bothclasses-(Class | Deriveclass)

The tool first read the user defined mapping specifications, and then map the mapping entities to the specified source code to abstract the system. The tool allows the user to use the regular expressions and define the abstract regular expressions at higher levels in the mapping to easily define the map entities for a particular task and details.

The tool also provides the integration mechanism through different components; a user can also extend the functionality by adding scripts and tools for extraction and abstraction purpose in the recovery process. The tool is applicable to large and different types of systems. It also supports the different recovery types (i.e. partial) of processes. For example, it can be applied to extract only particular artifacts from the subject system and irrelevant details can be ignored, or can be applied incrementally for investigation purpose for high level models recovery.

9. Conclusion

The software maintenance activities use the high level models to plan, design and execute maintenance tasks. The paper presents an approach to develop the high level model using the available documents, source code, experience and knowledge about the domain and application. The entities of the high level model associate the physical (files and directories) and conceptual associations to the source code. The physical and conceptual associations are represented by conceptual view, physical view and relational view, which help to represent the system at higher levels of abstraction. The Naive Bayesian classifiers is used to predict the entity association with the source code using information extracted from the source code and documents, experience and knowledge about the domain.

References

[1] Murphy, G., Notkin, D., and Sullivan, K., Software Reflexion Models: Bridging the Gap between Design and Implementation. *IEEE Transaction on Software Engineering*. Vol. 27. No 4: April, 2002, pp. 364-380.

[2] Nadim Asif, M. Dixon, J. Finlay and G. Coxhead, Recover the Design Artifacts. *Proceedings of International Conference of Information and Knowledge Engineering (IKE02)*, 24th –27th June, 2002 Las Vegas, Nevada, USA, CSREA Press, pp. 656-662.

[3] Nadim Asif, Reverse Engineering Methodology to Recover the Design Artifacts: A Case Study. *Proceedings of International Conference of Software Engineering Research and Practice (SERP03)*, 23rd-26th June, 2003, Las Vegas, USA,CSREA Press, pp.932-938.

[4] Nadim Asif, Muthu Ramachandran, Recover the Use Case Models. *Proceedings of International Conference of Software Engineering Research and Practice (SERP05)*, 27th -30th June, 2005, Las Vegas, USA,CSREA Press.

[5] Nadim Asif, *Software Reverse Engineering*, SoftResearch Press, 2006. (ISBN : 969-9062-00-2).

[6] Nadim Asif, Artifacts Recovery at Different levels of Abstraction. *Information Technology Journal*, 7(1), 2008, pp. 1-15.

[7] Nadim Asif, Faisal Shahzad, Najia Saher, Rafaquet Kazami,Waseem Nazar. A Case Study of Clustering the Source Code. *Computer and Simulation in Modern Science*, WSEAS Press USA. 2009. ISSN: 1790-2769. ISBN: 978-960-474-117-5.

[8] Nadim Asif, Recovery of Artifacts. *International Journal of Software Engineering*, Vol. 2, No. 1, pp. 11-16, 2008.

[9] Ghulam Rasool, Nadim Asif, Software Architecture Recovery. *International Journal of Computer, Information, and Systems Science, and Engineering*. Vol.1, No. 3, 2007.

[10] Ghulam Rasool and Nadim Asif, Design Recovery Tool. *International Journal of Software Engineering*, Vol. 1, No. 1, pp 67-72, 2007.