Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 2, February 2015, pg.389 – 394

RESEARCH ARTICLE

Categorizing Software Applications

Yogesh P Patil[1], N.D.Kale[2]

[1]Comp.Dept, PVPIT, Pune, India

[2]Comp.Dept, PVPIT, Pune, India

[1] yogeshppatil@gmail.com, [2] navnath1577@yahoo.co.in

Abstract— This paper provides an extensive overview of existing research in the field of categorizing software applications. This research is compared and discussed based on a number of different criteria: the categorization attributes that are supported, the specific techniques and algorithms that are used for supporting these attribute based categorization; the types of software artifacts that are being categorized; the important issues that need to be taken into account when building categorization engine and the use of software categorization engine. A running example is used throughout the paper to explain and illustrate the main concepts. The Microsoft's Windows AppLocker is feature for providing more flexibility for allow or blocking of application according to rules. But still there is scope to improve this system. Microsoft's AppLocker is supported only for specific OS Version like enterprise and professional versions. Also AppLocker works on Hash Based rules or Publisher signing certificate rules. So the applications not having valid sign certificate and get changed or updated is not feasible to block with the help of AppLocker. For such conditions of update of software IT administrators need to monitor continuously. So to overcome the problems in the AppLocker we are going to implement the Category based application engine.

Keywords— Hash, Publisher, Signing, AppLocker, Version

I. INTRODUCTION

Collin McMillan proposed the automatic categorization of software applications by using the criteria of API used by the application [1]. Our work is related to providing a solution for using software categories for the Category based application engine. Automatic categorization of software applications in repositories is increasingly gaining acceptance since it reduces the manual effort significantly [2, 4, 6, 8, 9, 10, 11, 12].Currently, software applications are categorized by applying text classification approaches; terms (i.e., words in identifiers and comments) are extracted from the source code of applications and these terms, also called attributes of the applications, serve as the input to a machine learning algorithm that eventually places applications into categories.

Another idea is to use external Application Programming Interface (API) calls from third-party libraries and packages that are invoked in software applications (e.g., the Java Development Kit (JDK)) as a set of attributes for categorization. Doing so is based on the fact that programmers typically build software using API calls from well-defined and widely used libraries [5]. The intuition behind our approach is that APIs are already grouped in packages and libraries based on their functionalities, and this grouping can be combined with machine-learning approaches to categorize applications. For example, a music player application is more likely than a text editor to use a sound output library, and finding APIs from this library in the music player application enables us to put it in a proper category. Moreover, APIs are common to many software programs and invocations of the API calls can be extracted from the executable form of applications because the API calls exist in external packages and libraries. Software repositories hold applications that are often categorized to improve the effectiveness of various maintenance tasks. Properly categorized applications allow stakeholders to identify requirements related to their applications and predict maintenance problems in software projects. Unfortunately, for different legal and organizational reasons the source code is often not available, thus making it difficult to automatically categorize binary executables of software applications. In this paper, authors proposed a novel approach in which Application Programming Interface (API) calls from third-party libraries are used as attributes for automatic categorization of software applications that use these API calls. API calls can be extracted from source code and more importantly, from the byte-code of applications, thus making automatic categorization approaches applicable to closed source repositories. [1]

Our work is related to creation of Module with combination of Software Restriction policy (XP) and AppLocker (window 7) to block or allow the applications according to specific categories of the software applications. Although software restriction policies and AppLocker have the same goal, AppLocker is a complete revision of software restriction policies. AppLocker was introduced with Windows 7 and Windows Server 2008 R2. You cannot use AppLocker to manage the software restriction policy settings. AppLocker rules are enforced only on computers that are running Windows 7 Ultimate and Enterprise editions or all editions of Windows Server 2008 R2, whereas the software restriction policy rules are enforced on these and earlier versions. Additionally, if AppLocker and the software restriction policy settings are configured in the same Group Policy object (GPO), only the AppLocker settings are enforced on the computers that are running Windows 7 and Windows Server 2008 R2. Therefore, IT administrators must use both software restriction policies and AppLocker in the organization, It is recommended to create AppLocker rules for computers that can use AppLocker policy and software restriction policy rules for computers that are running earlier versions of Windows. [14]

II. **OUR APPROACH**

The Category based application engine is a simple and flexible mechanism that will allow administrators to specify exactly what is allowed to run in their desktop environment. Category based application engine will provide a simple and powerful structure through one of the two rule actions allow or deny. It will also provide a means to identify exceptions to those actions. When Allow action is applied it will limit execution of applications to an allowed list of applications and all other applications will get blocked. When Deny action is applied it will take the opposite approach and will allow the execution of all application except only the applications which are listed in denied applications list. But sometimes will combination of allow and deny actions can be used. The ideal Category based application engine deployment uses allow actions on rules with built-in exceptions. Exception rules are useful to exclude files from an allow or deny action on a rule. Using exceptions, one can able to create a rule to "allow everything in the Windows operating system to run, except the built-in games." Using the allow action on rules with exceptions provides a robust way to build an allowed list of applications without having to create an inordinate number of policies. Category based application engine will support multiple, independently configurable policies for rules collections. In Policy the user can create the multiple allow or block rules providing greater flexibility and enhanced protection. For example, an organization could create a rule to allow the Software Developer group to run the installer or application from Microsoft for Visual Studio as long as it is still Visual Studio 2008.This will allow IT administrators to retain control but allow users to keep their computers up to date based upon their business needs. In addition, each of these policies can be individually placed into an audit mode. Audit mode is useful to test the rules before they start blocking applications from running and potentially affecting user productivity.

Category based application engine will provide following features for administrator perspectives for a corporate networks.

- Prevent unlicensed software from running in the desktop environment if the software is not on the allowed list.
- Prevent vulnerable, unauthorized applications from running in the desktop environment, including malware
- Stop users from running applications that needlessly consume network bandwidth or otherwise affect the enterprise computing environment
- Prevent users from running applications that destabilize their desktop environment and increase help desk support costs
- Provide more options for effective desktop configuration management
- Allow users to run approved applications and software updates based upon policies while preserving the requirement that only users with administrative credentials can install or run applications and software updates
- Help to ensure that the desktop environment is in compliance with corporate policies and industry regulations

Category based application engine will provide different built-in categories of applications for allow or block or allow the applications. The different categories that category based application engine will provide can be as follows.

- Instant Messengers
- Music Video Players
- Computer Games
- P2P Applications

III. MATHEMATICAL MODEL

We provide the following formulas for the reproducibility of our approach. Entropy is a measure of the uncertainty associated with an event and is expressed in terms of a discrete set of probabilities.
Pr(X) over an event $x_i \epsilon X$, where X is the event space

$$e(X) = -\sum_{i=1}^{n} \Pr(x_i) \log(\Pr(x_i))$$

Let C be the event indicating whether an application is a member of the specified category (e.g. if the application is related to the category). Let a denote the event that the software application contains the specified pattern attribute.

$$\Pr(C) = \frac{\text{number Of Related Applications}}{\text{number Of apllcaitions}}$$

$$\Pr(\bar{C}) = 1 - \Pr(C)$$

$$\Pr(a) = \frac{\text{number of Applications with PatternAttributeA}}{\text{number Of apllcaitions}}$$

Pr(C) is the probability, for each category, that an application will be in that category. Pr(a) is the probability, for each attribute, that an application will contain that attribute.

$$\Pr(\bar{a}) = 1 - \Pr(a)$$

$$\Pr(C \mid a) = \frac{\text{number of relate Applications with PatternAttributeA}}{\text{number Of apllcaitions with Pattern AttirbuteA}}$$

$$\Pr(\bar{C} \mid a) = 1 - \Pr(C \mid a)$$

$$\Pr(C \mid \bar{a}) = \frac{\text{number of Apps without PatternAttributeA}}{\text{number Of apps without Pattern AttirbuteA}}$$

$$\Pr(\bar{C} \mid \bar{a}) = 1 - \Pr(C \mid \bar{a})$$

The prior entropy represents the overall distribution of applications into a category, and is calculated as:

$$e(C) = -\Pr(C)\log(\Pr(C)) - \Pr(\bar{C})\log(\Pr(\bar{C}))$$

The posterior entropy represents probability of a given pattern attribute for a given category:

$$e_a(C) = -\Pr(C \mid a)\log(\Pr(C \mid a)) - \Pr(\bar{C} \mid a)\log(\Pr(\bar{C} \mid a))$$

Likewise, the posterior entropy of the category when the pattern attribute is absent is :

$$e_{\bar{a}}(C) = -\Pr(C \mid \bar{a})\log(\Pr(C \mid \bar{a})) - \Pr(\bar{C} \mid \bar{a})\log(\Pr(\bar{C} \mid \bar{a}))$$

Thus the expected posterior entropy is

$$EPE(C,a) = e_a(C)\Pr(a) + e_{\bar{a}}(C)\Pr(\bar{a})$$

And the expected entropy loss is

$$EEL(C,a) = e(C) - EPE(C,a)$$

IV. **ALGORITHMIC STRATEGY**

In this section, we present algorithmic details of our proposed methods. First, we describe In String Matching Algorithms we try to find the position where patterns are found within a larger string or text. String matching can be performed in the Text String through Single Pattern and Multiple Pattern occurrences. Multiple pattern matching provides solution concept in many applications. The Aho–Corasick is one of the string matching algorithms, invented by Alfred V. Aho and Margaret J. Corasick. For finding Multi pattern occurrences in the text string this algorithm is more appropriate because it performs exact matching of the patterns in the text. It seems to be like dictionary-matching algorithm which starts finding pattern on the basis of sub-string matching, each time character of pattern string is read and it tries to find the transition of that character in the already constructed automata, after reading the whole pattern string if the automata found to be entered in the final state so the pattern occurrence will be reported. Similarly it matches all patterns simultaneously. This algorithm can be applied to solve various problems like intrusion detection, detecting plagiarism, bioinformatics, digital forensic and text mining etc. Intrusion Detection is a technique in which intrusions are detected by Intrusion Detection System (IDS).Plagiarism Detection is process of finding plagiarism within a work or document. Bioinformatics is the application of computer technology to the management of biological information. Digital Forensic is a method for retrieving information from digital devices after being processed and generates some result. Text mining or Text Data Mining is the process that attempts to discover patterns in large data sets.[2,3,5,6,7].

Aho Corasick Algorithm:

Aho Corasick is the Multi pattern matching algorithm which locates all the occurrence of set of patterns in a text of string. It first creates deterministic finite automata for all the predefined patterns and then by using automaton, it processes a text in a single pass. It Consists of constructing a finite state pattern matching automata from the patterns and then using the pattern matching automata to process the text string in a single pass.[9,10].

Algorithm:

```
for each pattern Pat=1 to N
AddToTheTrie ();
end for
for each Version V= 1 to M
SearchAhoCorasickMultiple ();
end for
Above algorithm is used to search the patterns
```

Aho-Corasick (ACk) Phase 1:

Example: Suppose we have a finite set of patterns {FLVPLAYER, WMPLAYER, WOW AND SKYPE}. ACk algorithm in first phase creates finite automata for set of given patterns.

Automata for the Patterns Set= {FLVPLAYER, WMPLAYER, WOW AND SKYPE} is given below

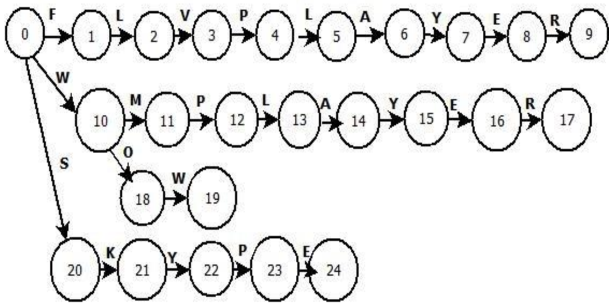

Fig 1: Automata

FAILURE FUNCTION:

We have considered Failure function which is defined as the longest suffix of the string that necessarily be the prefix of some node. The main goal of this failure function is to allow the algorithm not to scan any character more than once.

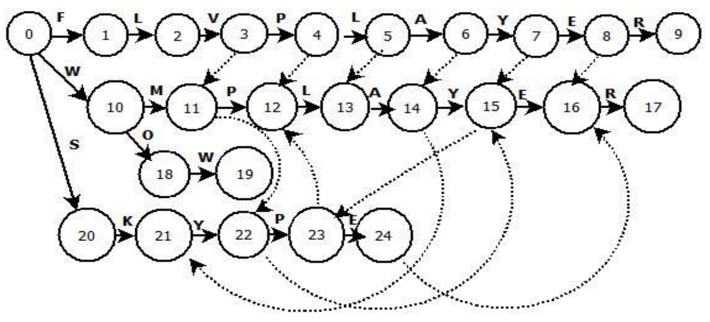

Fig. 2 Failure function transitions

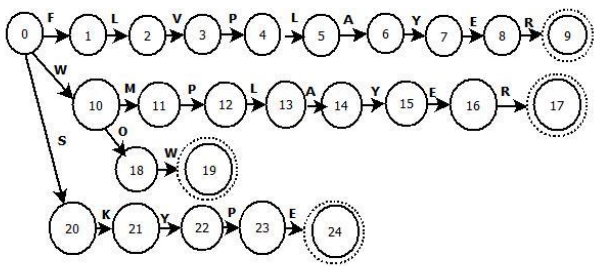

Figure 3: Output Function transitions

V. **RESULT SET AND DATA SET**

In this experiment we studied different algorithms and techniques used for categorizing software applications. We also studied different algorithms for pattern matching. We also compared those algorithms. We concentrated on time complexity in terms of time take to match the pattern. Our main concern will be to search and match given input buffer with predefined pattern database.

VI. **CONCLUSION**

After the detailed study of existing techniques in software categorization, we got to know that different techniques like API are used for categorization. We also studied the AppLocker feature provided in Microsoft in different Operating System. Our category based application engine will cover the drawbacks present in the Microsoft's AppLocker. Category based application engine system will support all versions of Windows OS. In AppLocker there is need of trained IT administrators to configure or create rules for blocking or allow of applications but Category based application engine will provide ready options for blocking most of the popular software categories like Instant Messengers, Music Video Players, Computer Games, and P2P Applications. So each time in each network the IT Administrators have no need to create specific rules for the applications.

ACKNOWLEDGEMENT

We would like to express our gratitude and sincere regards to the following people to whom we are grateful for their support and help. We would like to thank Dr**. Y.V. Chavan**, **Principal, P.V.P.I.T.**for providing us with excellent facilities and valuable guidance. We would like to express my profound and sincere gratitude to **Prof. N. D. Kale, HOD, Computer Department, PVPIT** who has guided me and given all his co- operation to complete this paper. We would like to express my profound and sincere gratitude to **Prof. Y. B. Gurav, Computer Department, PVPIT** who has given all co-operations and help to complete this paper.

REFERENCES

[1] Collin McMillan, Mario Linares-Vásquez, Denys Poshyvanyk, Mark Grechanik "Categorizing Software Applications for Maintenance" 2011 27th IEEE International Conference on Software Maintenance

[2] M. Bruno, G. Canfora, M. Di Penta, and R. Scognamiglio, "An Approach to support Web Service Classification and Annotation," in EEE'05.

[3] Thomas H Corman, Charles E. Leiserson, Ronald L.Rivest& Clifford Stein "Introduction to AlgorithmsString matching", IEEE Edition, 2nd Edition, Page no .906-907.

[4] G. A. Di Lucca, M. Di Penta,and S. Gradara, "An Approach to Classify Software Maintenance Requests," in ICSM'02.

[5] M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S.Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi, "An Empirical Investigation into a Large-Scale Java Open Source Code Repository," in ESEM '10.

[6] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, Mobasher B., Castro-Herrera C., and M. M., "On-demand Feature Recommendations derived from Mining Public Product Descriptions," in ICSE'11.

[7] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in CASCON'08.

[8] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Automatic Categorization Algorithm for Evolvable Software Archive," in IWPSE'03.

[9] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "MUDABlue: An automatic categorization system for Open Source repositories," JSS, vol. 79, pp. 939-953, 2006.

[10] P. S. Sandhu, J. Singh, and H. Singh, "Approaches for Categorization of Reusable Software Components," Journal of Computer Science, vol. 3, pp. 266-273, 2007.

[11] K. Tian, M. Revelle, and D. Poshyvanyk, "Using Latent Dirichlet Allocation for Automatic Categorization of Software," in MSR'09.

[12] S. Ugurel, R. Krovetz, C. Lee Giles 'z, D. M. Pennock, E. J. Glover, and H. Zha,"What's the code? automatic classification of source code archives," in SIGKDD'02.

[13] http://support.microsoft.com/kb/310791

[14] http://www.wikipedia.org