Reengineering Legacy to Modern (RL2M) System with One Time Checker (OTC) for Information System Evolution

P. SUDHAKAR, P.SAKTHIVEL
Department of Electronics and Communication Engineering
Anna University
Chennai, Tamilnadu
INDIA
E-mail: kar.sudha@gmail.com

Abstract:- The prime focus of the information system evolution process is the aggrandize productivity and quality of the various components of the system. The evolution process is always challenging as it leads to an increase in overall complexity especially when the system changes are mostly confined to part of it. In this respect to improve efficiency and decrease complexity, we propose a Reengineering model namely Reengineering Legacy to Modern (RL2M) system. In this proposed work, reengineering technique is implied to demonstrate how modern system can be obtained by converting a legacy system or application. This approach is developed to impose the dynamic program slicing as a method, which is basically used for simplifying programs by focusing on selected aspects of semantics. It also influences the value of the variable occurrence for a specific program input. The intermediate outcome of RL2M is to compute the dynamic slices for the legacy system. The obtained slices would be converted to a new system which is further integrated to a Very Large Scale Integrated (VLSI) application. In several VLSI applications, the integration could be more tedious by mapping the entire system components. In this proposed approach, a wrapper is created which acts as a common interface that would be linked with the legacy system for effective conversion. During reengineering, all the legacy systems are not compatible with the new system, which leads to inaccuracy. To avoid the issues of reengineering, we propose a method named One Time Checker (OTC) for legacy system conversion. Before the implementation of the migrated system, the converting system enters into OTC which is easily integrated with any reengineering approach. The main advantage of this proposed work is the OTC can be integrated with any reengineering process and it is virtual to end user with respect to the application.

Key-words: - System evolution, legacy system, re engineering, dynamic slicing, wrapper, VLSI.

1 Introduction

Legacy systems [1] are old computer systems that is continuously be used as it still functions for the user requirements. A software program in the legacy system is expected to perform for many years and undergo frequent updates and changes. Constant changes to such legacy software systems are always expected to face some quality issues. A way to eradicate those problems in the legacy system is Reengineering. The reengineering is a method of analysis and gaining comprehensive knowledge from the existing system so as to rebuild the source code, according to the new system requirements. During the new system development process, programmers have higher level of abstraction in statement level, usually with the help of an application development environment. This paper describes an implementation of a wrapper that can be used to translate/rehost legacy programs into the new systems. A good reengineering [2] approach should hide the complexity of legacy system. It acts as an interface between the applications. When the system is reusable then the system components shared by the new system so the repetition is avoided. If these components are too complex then it is difficult to modify and extract the information to new system. The cost, platform and time are the main features that decide that the older system is easy to maintain or to redevelop or redesign the system to modern one.

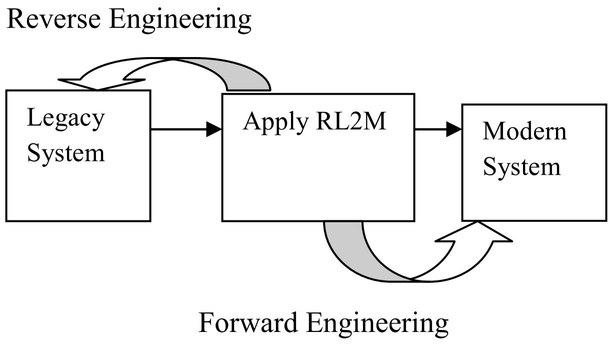


Fig 1.1 RL2M System Overview

Major researchers are still working with systems that were designed and implemented long ago for various organizations. Those systems depend upon some older technologies and their platforms would not provide easy support to the users. Moreover it is very difficult to add more functionality in a legacy system and also too complex to recover from situations like hardware failure, disaster etc., and Legacy system needs continuous maintenance which requires significant cost. The prices of the new system are fallen so much recently and the things are feasible because of wide availability of components. By considering these factors, it is essential to modify or transform the legacy system to the modern environment. It is about retaining and extending the value of the legacy investment through migration to new platforms.The process of re-organising a system so that related components are collected together in a single module. Usually a manual process that is carried out by system inspection and re-organisation. The purpose of evolution is to provide change, propagation and impact analysis.

The above diagram Fig 1.1 gives the simple process activities in the overview of RL2M. The translation of legacy to improved system is called forward engineering and vice versa. In reengineering, it is tedious to assure that the changes made in the legacy system will not introduce any bugs in the migrating system. The existing system is typically structured into multiple components, each consisting of hundreds of Lines of Code (LOC) and components. Before migrating the source code to target code, we should have at least a partial understanding of existing software. The existing systems are often hard to maintain, improve and expand is the reason behind the migration. The basic reengineering tasks are preparation for functional enhancement, improve maintainability, migration, improve reliability etc., It makes the software easier to change and improve its reusability. The ultimate thing of Reengineering is to keep the older system as it is and add the new things to it that leads to target system.

Migrating to a new platform leads to align applications with current and future needs through the addition of functionality and through application restructuring. A process of converting a predetermined code to another with a same or different code is called code conversion. The code conversion process consists of a number of specific steps as follows: Design model target programming system after analyzing legacy programming system and its attributes, develop migration model and associated procedures between legacy and target systems. Convert the legacy code to modern language through compiling/simulating the new test program, refine models to correct anomalies, address, omissions, and include upgrades and repeat these process until desired level of migration is achieved. In RL2M, the program slicing [6, 7] is used as the debugging technique in the system extraction process. The program slicing is a well established technique for program understanding and comprehension. In order to increase the precision of the program slicing, refine the program slicing with dynamic point of data which is known as dynamic slicing [7, 8]. For Migration we use wrapping which surrounds the existing thing, individual and application system and acts as a interface with new operations. Hence, reengineering with wrapping [4] gives a new and improved look to a legacy system. If the legacy system is infeasible then the good solution is to wrap the legacy system. The time needed for wrapping is minimum comparing to redeveloping or restructuring which are the phases of reengineering.

Almost all the reengineering activities perform a successful transformation. But they failed to produce a perfect resultant system. Consider a legacy system is highly constrained then the transformation itself is a tedious one. After the transformation the resultant system will not provide user friendly system. More effort is requires to examine and alter the target system which is a time consuming process. The transformation is not possible for a large process and erroneous task oriented system. Though the transformation is successful, the updation to that target system in consideration of latest technologies is a tedious task. During reengineering, it is tedious to analyze the legacy system. The major problem faced in the reengineering activity is that one-part of the reengineering team have the knowledge on legacy systems and others have only in the target system [3, 4]. Based on the assumption, the transformation is made especially in the situations such as bigger size, time pressures, the exclusive functions of legacy and target systems etc., as they are incompatible with each other. In reengineering there are many possibilities for the incompatibility, inaccuracy moreover they are very contrast with each other. Our proposed RL2M also faces all those problems. For this purpose, we propose a method named One Time Checker (OTC) for legacy system conversion. As name implies OTC checks the target system once again for its user requirements satisfaction. Our proposed OTC is not built for RL2M it is only implemented in RL2M.OTC is easily integrated in any reengineering approach which enhances the reengineering process.

This paper is structured as follows:

In Section 3, we propose a slicing algorithm and a method for debugging and converting a legacy system to modern system. In section 4, we implement the proposed method with some C++ programs and give the experimental results. In section 5, we propose a method to enhance a reengineering approach. In section 6, that enhancing approach is implemented in our RL2M and the results are discussed. In section 7, we conclude our discussion with the presentation of further enhancement of our proposed approach.

2 Related Works

In the literature, only few things are available which address database reengineering and software quality. Manual translation of legacy to modern system is the most common approach. In this approach, two editors are required for legacy and modern system separately. The effectives of the manual translation process are determined by the degree to which the legacy code meets the compatibility considerations. It is impossible for migrating large programs using manual conversion such as VLSI. Another approach is an automated translation of legacy system to modern language. This approach performs rigorous analysis on a legacy system, providing detailed output on the changes required to the new system. This approach allows easy movement between legacy and modern language. But the effectiveness measured by means of technique and the effort put on that technique. Existing research work extracts the legacy system such as rules. Many approaches to the reuse and integration of legacy system took place in the previous works. In legacy system modernization, number of approaches is available and several techniques were proposed. Some of the work presents a generalized model of the software life cycle that recognizes explicitly the legacy system to the attainment of new system from reusable components.

3 RL2M System Architecture

In this paper, we propose a new model called Reengineering Legacy to Modern system (RL2M) for abstracting source code from legacy system. This approach involves the functionalities: (1) Program analysis (2) Identification of the constructs from the legacy system (3) Slicing of the code (4) Mapping source and destination program to create template (5) Creating a wrapper (6) Execution (7) Integrate to VLSI application. The RL2M architecture is shown in the figure Fig 3.1. Initially, the source program is analyzed that implies the inventory of all applications "artifacts". In this step, we have to analyze all the components of the legacy system including tables, views, indexes, data profiling. In the constructs identification, the legacy program is taken with various constructs. Then the program with required constructs is categorized according to control flow activities. It also enables to check whether the legacy system follow the syntax and semantic. The primary requirements for these construct identifications the input source program should be executable in the desired environment. These programs are categorized depending upon the constructs. If all the requirements are satisfied, then the legacy program is compiled that yields a batch file for the program slicing.

For the given input program the slices are computed for obtaining the program slice of smaller size or of equal size in the worst case for the particular extension of the program. Dynamic slicing is one of the program slicing techniques where the source program decomposes and produces the slices.

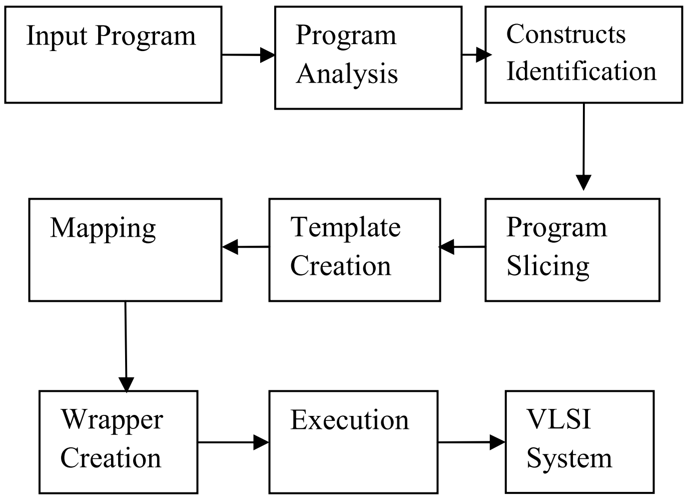


Fig 3.1 RL2M Architecture diagram

The input program contain faults in which program slicing determine those statements and the failure has been revealed for a given input .It finds all the statements in a legacy system that affects the value of the variable occurrence. It is used for simplifying programs by focusing on selected aspects of semantics. It consists of statements that influence the value of the variable occurrence for a specific program input. It also distinguish between multiple occurrences of the same instruction .Program slicing has the ability to assist in tedious and error prone tasks .In RL2M a novel algorithm is presented below to perform a program slicing, This algorithm initially get the input program and defines the slicing criteria in which the slicing variable is initialized. The algorithm further splits a program and checks the program for errors. This algorithm checks the input program without tedious computational complexity also with less time.

3.1 Algorithm for Program Slicing

```
1. Get the Input program to be sliced
2. Define the slicing criteria ( $s$ , $v$) where $s$ = statement number, $v$ = slicing variable
3. Check whether the slicing variable $v$ is present in statement number $s$ in the Input program
4. If $v$ is available in $s$, go to step 5, else escape line $l$ and continue
5. Let $k$ = $v$; count_line = 0 ; array found_variables[] = $v$
6. For each statement line $l$ in Input program statements
   {count_line = count_line + 1
        If variable $k$ is present in $l$ then
        { move the statement line $k$ to an array found[] }}
7. line_no = 0
8. Let converted [] to store converted code, st as statement, cmt as comment and fn as function
9. For each statement line $l$ in array found[]
   {Let variable $k$ = found [line_no]
        If variable $k$ = st then
        {Convert $k$ and Add to array converted [] }
        If variable $k$ = fn then
        {Convert $k$ and Add to array converted [] }
        If variable $k$ = cmt then
        {Add $k$ to array converted []      }
        line_no = line_no + 1 }
10. Display array converted[]
```

The application of this algorithm are architecture reconstruction, identify reusable functions, program comprehension, debugging and maintenance. Hence the flaws are relieved from legacy systems. If all the requirements are satisfied for the computed slices, then the RL2M system will create a corresponding file that contains the main function of a target system as a template. A template method defines the program skeleton of an algorithm. The mapping is a collection of objects that specifies the transformations that are required to migrate a part of the legacy system. The dynamically sliced construct of the source program and the obtained slice are mapped to the destination program .It is sophisticated to the user while the proper name given for the template because the destination target system attain the output position. This is necessary that the sliced output is again converted to the source code rather than contain the instructions number alone and avoid overlapping or collisions between the legacy systems.

Wrapper is a popular software component that converts a system from one version to another. Legacy systems can be used on various models ranging from standard file structures to relational and data models. To deal with this heterogeneity, a wrapper must hide the model that a legacy system implements by providing a more and common model e.g. a canonical model that is highly generic and more flexible than the legacy systems. Wrapping surrounds the existing data, individual program, application system and interfaces with new operations. Hence wrapping gives a new and improved look which allows reusing legacy components. This component helps in complex problems which unlocks the value of the legacy input and open a new solution to rebuild the legacy system. In RL2M it actually includes a new source to a legacy system by act as a interface between them. When the legacy system are kept unchanged, while the new component is designed and developed by the modern practices. Wrappers are used to extract, update and control the implicit constructs of the legacy system by the preceding steps. Wrappers provide robustness and also deliver a target system in a effective manner to the user. It is typically encompasses a combination of other process such as reverse engineering, restructuring and forward engineering.

Execution is the process by which a computer or a virtual machine carries out the instructions of a computer program of new application. The instructions in the program trigger sequences of simple actions on the executing machine. Those actions produce effects according to the semantics of the instructions in the program. The migrated code will be used in a VLSI application where Integrated Circuit (IC) contains millions of transistors, each a few mm in size for a specific function. Because of wide ranging of application, the destination program is applied in VLSI application. It is laborious since it has characteristics such as process variation, stricter design rules; first-pass success etc., Integration of VLSI sometimes leads to serious effects which are avoided by OTC. To convert a legacy system to modern system is the RL2M first preprocess the input legacy system for their functions and then convert the functions to the modern system.

Each of the above mentioned process in the RL2M has got their own importance. The assumptions made are that during the slicing of the required constructs it is necessary in getting back the input program and should ensure efficiency and accuracy of target system. Hence, each process is in turn dependent on each other. The entire RL2M is going to be sequential. No process can be give priority. The RL2M has to be carried out sequentially not simultaneously.

4 RL2M: Experiments and Results

We have conducted an experiment by using RL2M model to evaluate how program slicing works and the source code converted to modern language. We considered a partially Object Oriented (OO) language like C++ as the legacy language. Though it supports several OO features, it has some limitations such as security constraint due to usage of pointers. It does not support network interface and hence cannot be used in web based projects. Unlike Java it is also not platform independent. OO programming has many positive aspects over the non object oriented. Many legacy systems were developed before the OO programming concept. Most reengineering activities focus on the functionality of the original program, and the OO redesign results entirely new in which only the designer understand the original program. These are not sufficient as they not only take more time and also required more effort for designers besides it is mistake prone due to the human involvement. Hence conversion of non object oriented to object oriented language is the need of the hour. Our experiment dealt with this kind of conversion to convert the partially OO code to purely OO code. It is infeasible to convert C++ to Java also it is too costly and time consuming process while redeveloping. But in our RL2M the wrapping takes place for conversion it leaves the code in current environment and connects to a new environment with a minimum change to legacy system. RL2M hides the legacy C++ program and performs the execution of Java. The process of our proposed RL2M is explained below with suitable examples.

RL2M gets the legacy input C++ program first, and then the processor analyzes the statements of the legacy code. It also identifies all the input statements so as to replace it by their equivalent Java program. Header files and access specifiers are inserted in the template. The branching and looping statements are similar in both the legacy and new application but for planning each and every statement into java file, these simple constructs are identified and the conversion takes place. For object identification the character is checked to be either object or a variable. If it is a variable the data types, if it is an object they can be created only with class name which for a user defined data type. Hence the object identified and replaced with new operator in the java statements. Hence the objects are identified to the class objects and not variables and replacing the operator with thus performing the require operation. The operators +,- and * are converted in RL2M as operatorMinus, operatoPlus, operatorMul respectively. Any other normal arithmetic operations are to be left as such without converting in this approach.

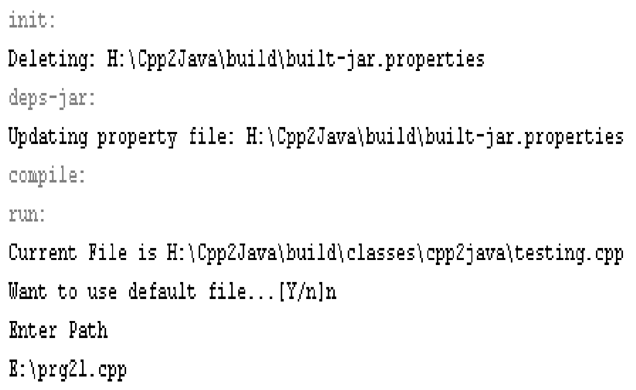
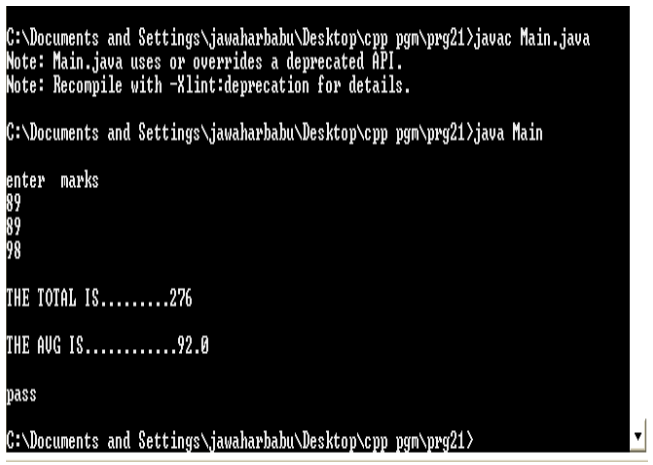

```
init:
Deleting: H:\Cpp2Java\build\built-jar.properties
deps-jar:
Updating property file: H:\Cpp2Java\build\built-jar.properties
compile:
run:
Current File is H:\Cpp2Java\build\classes\cpp2java\testing.cpp
Want to use default file...[Y/n]n
Enter Path
E:\prg21.cpp
```

Fig 5.1 Retrieving a CPP program as Input

The above figure Fig 5.1 shows a step of retrieving a CPP program which would be converted to Java. The CPP program is collected from the computer system by entering the path it has been stored in the system. After the input is given to RL2M, we have to run the RL2M model so as to convert the input C++ to output Java. For this retrieving step we must have a collection of C++ programs with various constructs. These programs are categorized depending upon the constructs. The C++ program may or may not contain the classes by default. This process analyzes the program for the construct type that undergoes the slicing algorithm to identify the flaws in the taken C++ program,


```
C:\Documents and Settings\jawaharbabu\Desktop\cpp pgm\prg21>javac Main.java
Note: Main.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Documents and Settings\jawaharbabu\Desktop\cpp pgm\prg21>java Main

enter  marks
89
89
98

THE TOTAL IS.........276

THE AVG IS...........92.0

pass

C:\Documents and Settings\jawaharbabu\Desktop\cpp pgm\prg21>
```

Fig 5.2 A successful conversion of CPP

The above figure Fig 5.2 gives the successful conversion of CPP to Java. The converted program which will be stored in the directory where the input CPP is stored. The output Java is viewed on Java editor and can be compiled in that same editor.. After compilation, a Java class is created for the CPP input and the class content corresponding to the functions and statements, which are to be mapped later. The main program contains the structure of main with the main method in which the functions from class program can be called at runtime while mapping with file contents CPP into JAVA as output. The output of the slice is mapped to Java for creating a Java program to obtain the Java template of classes and main function. The experiments were conducted for more than 50 programs and the results are discussed below.

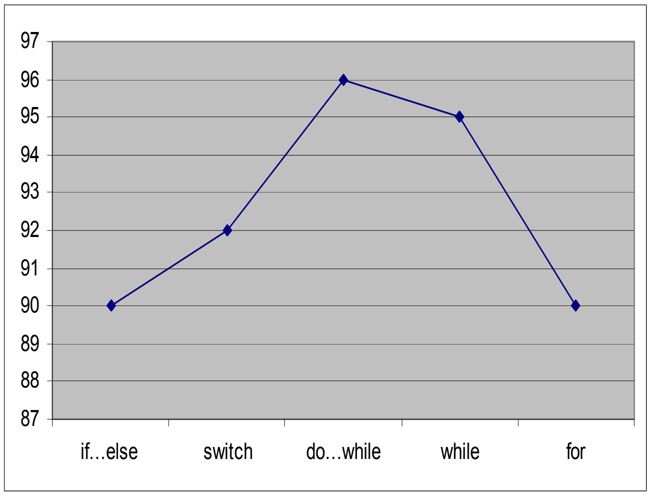


Fig 5.3 Percentage Efficiency in code conversion for control structures

The graph given above in Fig 5.3 shows the efficiency of the RL2M experiment that was conducted for various control structures and the performance of RL2M which yields significant performance in code migration. The various constructs in C++ is successfully converted to Java and as it yields a good efficiency. Our proposed method is evaluated based on the cost of execution and how they get to the desired accuracy. All of our results were generated from independent experiments and the results are averaged for further work. A static cause which allows errors to occur when the internal state of a program is invalid or a legacy program is invalid. If it is a case examine an input program to see what is supposed to do and what is not supposed to do. This type of inconvenience is reduced by means of OTC which is discussed later.

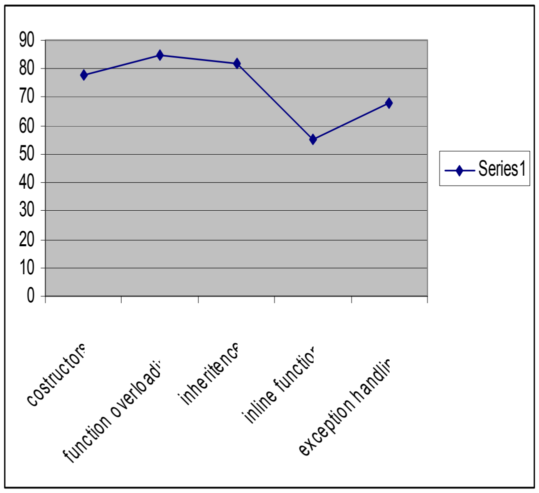


Fig 5.4 Percentage efficiency in code conversion for various OO concepts

The above graph in Fig 5.4 contains the main results of the experiment conducted and compares the performance of various OO concepts which undergoes RL2M technique. It yields a good result for constructors, inheritance, function overloading. But the migrating java not merely supported the concepts like inline functions. If we compile the legacy system as like inline functions or graphics it is tedious to programmer to compilation and execution and the human involvement is necessary. To eliminate these bugs it is necessary to check the target system before going to execution. This complexity is very common in all reengineering approaches. Apart from these the transformation is successfully done for constructs like overloading, conditional branching, iteration statements, arrays and compound statements. The number of incorrectly converted programs is sampled and sends for experiment after integrating OTC. There are numerous rules defined for these conversion and they were discussed below.

The major reasons for reengineering fails in the reengineering process itself because reengineering method did not avoid the contrast between the legacy system and target system and the human consideration is needed. The manual debugging process does not sophisticate for each and every process and it is impossible for process like VLSI. Our proposed RL2M efficiently handles this situation. For our taken examples it gives good results. Although it gives successful conversion it fails in input programs like inline functions, graphics etc. To overcome these drawbacks and to enhance all reengineering approach, we propose an automated technique name as OTC. This OTC helps to removes the bugs in target system successfully and enhances the approach easily.

5 OTC

The main challenge in any reengineering approach is to take legacy system and deliver a good translation methods and attributes, which leads to a new target system that keeps the older functionality while applying translation method without any serious defects. For this purpose, OTC is a sophisticated and valuable methodology for the reengineering. It applies a gradual process in a reengineering approach and produces a target system which satisfies the target system compatibility and requirements. As name implies OTC checks the target system once again for its user requirements satisfaction. It based on Meta programming. It is applied as pattern based generation and it is automated task.OTC mainly used in situations like hard to derive target system complicated transformations, VLSI integration, time consuming process etc.,

Each system has its own peculiar and exclusive function. All systems follow some unique patterns. Our aim is to define and identify that pattern before the implementation of the converted target system. This approach is automated wherein the manual conversion is also available as optional at the stage of execution of target system. Our approach is suggested for many business organizations as it captures the legacy system and represents them accurately. To overcome all the above mentioned drawbacks on every approach, we introduce a new method where the bug detection and rectification is done on target system. The basic requirements expected for OTC is a legacy system and the method for reengineering. The compiled legacy systems that were obtained by debugging techniques such as program slicing, breakpoints etc., undergoes a reengineering transformation process and produce a target system. Our method is to check and ensure that target system for their peculiarities and exclusive function. This proposed new method is named as One Time Checker (OTC) which takes place before the compilation and execution of target system after the reengineering process.

OTC provides very easy and user friendly conversion system that will support all the platforms. It also provides the integration among the resource sharing with the distributed environment along with the new technologies such as web services and traditional methods. In addition, our proposed component suit for different programming languages that would be able to communicate with the network also. The idea for this technique is quite simple based on a fact that each system has its own peculiar and exclusive function. In this OTC, tokenization will be performed which is a linear one. The set of delimiters which defaults to common whitespace characters may be specified at creation time or on a one token basis.

In this OTC, we have built-in functions which comprises of Libraries, Tokenization, Pattern matching and the special function contains the appropriate errors and their solutions. After the termination of reengineering process the resultant system is inserted into OTC then the target system is obtained. In this OTC flag is introduced when the flag is true then the resultant is ready for compilation and it is false then we ensure that the resultant system contains some serious errors or exceptions. When it leads to false then the human interaction may be needed for examination. For this purpose we develop a window which contains error and warnings. It is automated and produces almost accurate translation and the system is ready for compilation where the interference and incompatibility between legacy and target system are avoided. This OTC is possible to develop for all the systems and it is very easy to integrate with any reengineering approach. The manual compilation is minimized because of this automated error debugging task. The proposed process diagram is given below fig 5.1

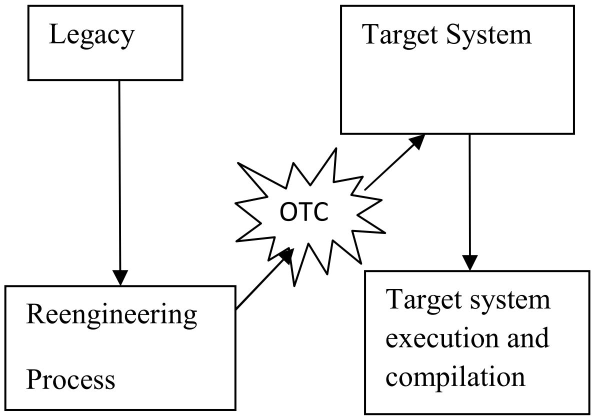


Fig 5.1 Process diagram of OTC

The OTC provides many modules which happen sequentially. The modules presented are (i) Tokenize (ii) Pattern Matching (iii) Debugging. Tokenize function breaks the code into tokens based on their creation time. It does not distinguish the legacy and target code but the libraries take care of that. Tokenization is a simpler task. Standardized and updated libraries provide a generic way to access the exclusively features of target system in pattern matching. At pattern matching the executed code is matched and sends for verification. All the system has a particular structure which is verified for pattern matching. In this rule based matching is performed. To successfully store and retrieving the executed program we provide a Hash table which contains a key which implement the method. OTC provides program debugging, testing, parallelization, integration, safety, understanding, maintenance and metrics and so it acts as power of reengineer approach. The primary goals of this OTC is to provide a simple , familiar, architecture neural, portable and high performance to end user, The module diagram of OTC is given below in Fig 5.2.

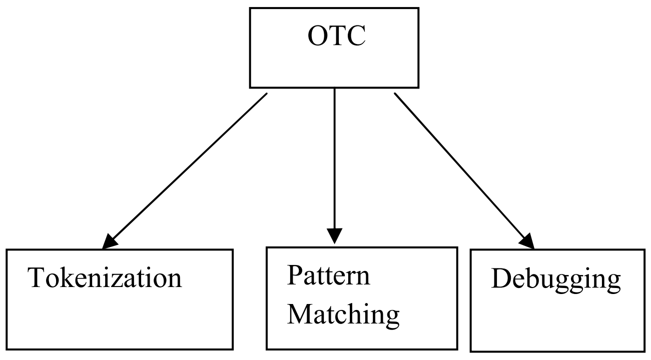


Fig 5.2 Module diagram of OTC

In debugging module, the most probable errors and their solutions are stored. This automated debugging point out the appropriate errors and replaces it with correct one. In this module we store and retrieve the possible error. There is no need for special computation technique for debugging. Hence, the target system compilation becomes easy. In this OTC there are some constraints such as it support only one terminal, and the information handled in OTC is text based. It is possible that more than one user can use this OTC at a time simultaneously but it is same as that of the reengineering approach.

When OTC is integrated in reengineering approach the main criteria to consider is time. It may take more than 12 seconds and less than 1 minute depends upon the task and the program input. Although it takes more time it completely reduce the programmer burden at the time of debugging. The OTC automatically replaces the abrupt errors also it carries many corrective measures. By considering these factors the time consumption of the proposed OTC is negligible also it equalized that time in compilation and debugging of the particular target system. The above described OTC and their time consumption properties are describe below with suitable examples.

6 OTC: Experiments and results

Experiments were conducted for our proposed system and the results discussed below. In our experiment in RL2M C++ act as legacy and the resultant is Java. Legacy source is inserted to the transformation approach and the OTC integrates with it. After reengineering, tokenization is the simple method its performance is based on creation time and a per-token basis. It breaks statements into tokens. A tokenization maintains a pointer which maintains a current position past the characters and it advances the current position. At the next module, the obtained tokens are matched with predefined inbuilt OTC libraries. In this, transformed statements scattered throughout the program for finding the irrelevant statements when it found irrelevant statements it takes automated corrective measures. The matching is made efficient by the use of some data structures search techniques.

We conducted the matching by hash table where searching and matching are sequentially. The OTC debugging is different from compilation and execution. This module omits some abrupt errors. In this module some appropriate errors stored and the corresponding solutions also given. There is another option where the ideas inserted as comment lines. Human interaction is not possible because it works virtually but it is available at the end of transformation where the flag became false if it is true it is ready for compilation.

OTC is the most powerful techniques to transform existiong syetm to modern system. It provides unique features like a detailed output on the changes necessary will make transformation a much more efficcient and reliable process. It is able to integrate on any reengineering process and there is no need to free up memory which are considered to be the significant benefits of OTC. As the experiment results conducted on OTC, it gives increased performance and reduced burden of target system compilation. The future work may be the extension of the transformation to highly tedious legacy systems with minimum time requirements. The process is much more efficient and give considereable improvement when the modules grouped together as a single unit in the target compiler itself. OTC takes a few times more for execution as it built in with reengineering approach. The following time considerations are explained below. The time consumed for OTC and non OTC process conducted on various programs are given in Table 6.1.

Attributes	OTC	Non-OTC
Arithmetic operators	38 sec	52 sec
Realtional operators	40 sec	75 sec
Compound assignments	52 sec	72 sec
Function calls	63 sec	75 sec
Bitwise operators	30 sec	56 sec

Table 6.1 Time consumed by various programs

This table concludes that the time needed to OTC is high than non OTC process but it is negligible on performance aspect of view because abrupt target system errors takes more time to compile also it is very tedious. The efficiency of OTC is further explained in the Fig 6.1. All the input is once again check in this OTC to ensure the target system functionalities. In our experiment the considered graphics functionalities input system is successfully traced by OTC and it suggested by the java functionalities such as AWT components. Then the user interaction made easier where the programmer complexity reduced considerably. The results of this OTC are discussed below with suitable examples. More than 50 programs were considered and there is good improvement with OTC. The proposed OTC is integrated in any reengineering approach so as to easily eliminate the target system bugs. The below Fig 6.1 explains the efficiency of systems with OTC and Non OTC process. In the first phase of OTC Java undergoes the tokenization after that the tokens patterns matched with inbuilt functions. After the debugging phase the equivalent Java code will generate as target system. In our experiment we provided a conversion of some main features such as call by reference, call by value, function overloading, inline functions and inheritance. It is powerful rule based matching technique. The efficiency may be improved by hash table.

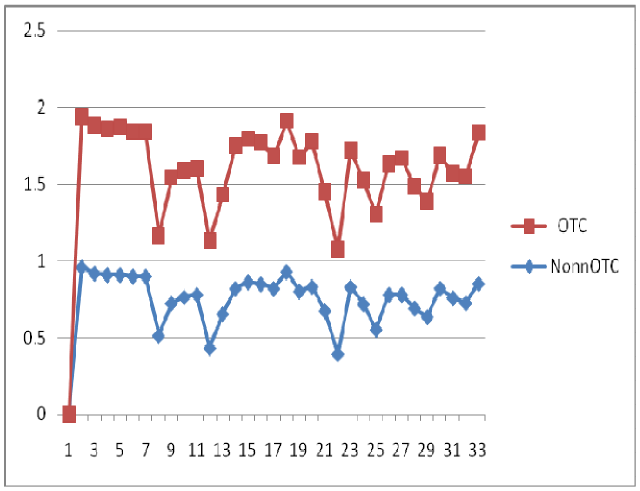


Fig 6.1 Efficiency on different programs for OTC Vs non OTC

The above graph shows the effeciency that was conducted for different programs and it ensures that OTC gives considerable improvement in performance and accuracy during trnsaformation. OTC provides detailed output on the changes necessary,will make transformation a much more efficcient and reliable process.It is able to integrate on any reengineering process and there is no need to free up memory which are the benefits of OTC. The future work proposes the extension of the transformation to highly tedious legacy systems with minimum time requirements.The process is much more efficient when the process executed in the target compiler itself.

7 Conclusion

In several applications the transformation of legacy system could be more difficult by mapping the entire source program into the modern one. In this approach, we created a common interface that can be linked with the legacy system which is dynamically sliced and results are obtained. The main advantage of applying dynamic slicing technique is, the source program and its components are identified with respect to a slicing criterion and the same is converted and verified after the migration. Since program slicing is a debugging by using the RL2M and OTC techniques, the migration can be done without any errors even with any run time inputs. The output of these two tools is mapped to Java for creating a Java program to obtain the Java template of classes and main function.

The Proposed RL2M gives a good conversion technique when compared to other techniques due to their semantics checkers and wrapping. As like other approaches RL2M have to faces some challenges especially situations like VLSI integration. OTC recognizes these issues and makes the conversion accurately for tedious and larger tasks. The future work proposes the extension of OTC and RL2M where the exclusive functions in legacy system are to be mapped and converted to the equivalent target system then to work as an executable target system itself without bugs.

References:

[1] Declan Good, “Legacy Transformation”, Technology Research Club (Club de Investigation Technological S.A), 33rd Research report, An Jose, Costa Rica, Aug 2002, pp.125-154.

[2] Dr.Linda H.Rosenberg, “Software Re-engineering", software Assurance Technology center, Goddard Space Flight Center, NASA, Technical Report, 1999. pp 67-109.

[3] H. M. Sneed, “Encapsulation of legacy software: A technique for reusing legacy software components”, Annals of Software Engineering, 9:293–313, 2009.

[4] Thiran, J.L. Hainaut, and G.J. Houben. “Database Wrappers Development: Towards Automatic Generation”, In Proceedings of the Conference on Software Maintenance and Reengineering, pages 207–216. IEEE Computer Society Press, 2005.

[5] Agarwal. H and Horgan. J, “Dynamic program slicing”, In SIGPLAN Notes no. 6, pp. 246-256, 1990.

[6] Durga Prasad mohapatra,Raib Mall and Rajeev kumar,” A Novel approach for computing dynamic slices of object-oriented programs with conditional statements”, IEEE 2004.

[7] Ati Jain, Swapnil soner, “An approach for Extracting Business Rules from Legacy C++ code,”, IEEE 2011.

[8] Chuanqi Tao, Bixin Li, Xiaobing Sun, Chongfeng Zhang, School of Computer Science and Engineering, SoutheastUniversity , Nanjing, China, 2010 34th Annual IEEE Computer Software and Applications Conference.

[9] Ritu BajpaiMona Zaghloul,Department of Electrical and Computer Engineering, The George Washington University, Washington DC, USA, 2008 IEEE.

[10] Csaba Farago, Tamas Gergely, “Handling pointers and unstructured statements in the forward computed dynamic slice algorithm”, Acta Cybernetica, v.15 n.4, p.489-508, December 2002.

[11] Wichaipanitch, W.; Samadzadeh, M.H.; Tangsripairoj, S, “Development and evaluation of a slicing-based C++ debugger”, International Conference on Information Technology: Coding and Computing, 2005. ITCC 2005. Volume 2, 4-6 April 2005 PP.473-478.

[12] Ashida, Y., Ohata, F. and Inoue, K.”Slicing Methods Using Static and Dynamic Information”, Proceedings of the 6th Asia Pacific Software Engineering Conference on Software Engineering, pp.509-518, Baltimore, Maryland, May 1993.

[13] Basili, V. (1993) “The Experimental Paradigm in Software Engineering”. In H. Dieter Rombach, V. R. Basili, & R. Selby (eds.), Experimental Software Engineering Issues: Critical Assessment and Future Directives. Proceedings of Dagstuhl-Workshop, September 1992, published by Springer- Verlag, #706, Lecture Notes in Computer Software.

[14] Alessandro Bianchi, Danilo Caivano, Giuseppe Visaggio, “Iterative Reengg of Legacy Systems”, IEEE Transactions on SE, Vol 29, No 3, 2003.

[15] Zhuopeng Zhang, Hongji Yang, William C.Chu, “Extracting Reusable Object-Oriented Legacy Code Segments with Combined Formal Concept Analysis and Slicing Techniques for Service Integration”, Proceedings of the Sixth International Conference on Quality Software, 2006.