International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Business Rule Inclusion and Legacy Modernization through Binary Instrumentation

Prasenjit Kundu[1], Dr. B. K. Ratha[2]

[1]Research Scholar, Utkal University, Bhubaneswar, India

[2]Reader, Dept. of CSA, Utkal University, Bhubaneswar, India

Abstract: ***Business rule is essential for decision making, events, and process activities. It is used to capture knowledge and develop business systems' requirements. Frequent changing scenarios in different aspects of business, demands corresponding runtime addition or modification of such logics to enterprise's legacy system without disturbing their normal executions, functions and nature. This is difficult from reverse engineering point of view as redesigning such system with added features may not be cost-effective, reliable and feasible in all circumstances. One of the promising approaches to do so is plug in instrumentation. In this paper, we discuss the object code based aspects of dynamic binary instrumentation to add new business logics to organisation's legacy system at runtime without disturbing operational tasks .We also discuss the merit and demerit of such plug-in.***

Keywords: Business Rule, Reverse Engineering, Instrumentation.

1. Introduction

Legacy system contained different business logics buried inside them in unstructured and scattered manner. Modern changing scenario of business fertility demands frequent changes in their business logics running on their software system without hampering routine business tasks. This is complex as such legacy modernization not only requires extracting existing logics from system but also required effective techniques to merge and attach new logics in existing system. Legacy modernization is a costly, complex and incremental development process which requires migrating towards new technology, platforms and features. The main aim of such modernization is to retain the value of the legacy asset on the new platform and to better adaptability through identify and change the rules in the runtime. These aids reverse engineering. Different tools and techniques has proposed by different researchers over decades including Instrumentation, which involves adding extra code to a software system for monitoring some program behaviour, statically (i.e., at compile time) or dynamically (i.e., at runtime). In this paper we focused on how new enterprises logics at runtime can be embedded in legacy system using dynamic analysis to make them future ready.

2. Literature Survey

Olegas Vasilecas [1] described Business rules as an important and integral part of information system by expressing business logic, constraints of concepts, and their interpretation and relationships. Hence, it is needed to pay special attention to business rules in development of information systems. Anis Charfi [2] applied the divide and conquer principle to web service composition by explicitly separating business rules from the process specification. Nicholas Zsifkov [3] says Enterprise business rules are usually defined as constraints or as metadata about business operations: on the business side, business rules are special policies that define constraints/metadata about the business operation; on the information system (implementation) side, business rules are constraints about the data, about data manipulation and about system processes. E. Putrycz [4] highlighted in his paper how such legacy system and business rule can be connected together which serving as a basis of many researches in this domain. Legacy system contains numerous such business critical rules embedded within them which are essential for routine business operations and critical decision making. But these rules are difficult to separate from running system as there are gap between initial design documentation to the current executable program as those program gone through several evaluation cycle over time. Program analysis is one of the techniques to diagnoses a program either at compile time (called Static program analysis) or at runtime (called dynamic program analysis) to visualize the business logics embedded in those programs using graphical means. Tarja [5] described the way to analyzing object oriented program by combining metrics and program visualization techniques. In last several years different program analysis techniques had been proposed by researchers for better understanding, verification the program and to extract enterprise logics extractions from such information systems. But modern dynamic business world demands not only to extracts previous rules from their legacy system but also ways to integrate new business critical logics with existing rules for better business processing. This is a real challenge to add new business policies with existing rules at runtime without disturbing routine operations of such software system.

One of the promising alternatives of achieving this is program instrumentation. Torsten Kempf [6] defined instrumentation as adding extra code to an application for monitoring some program behaviour, can be performed

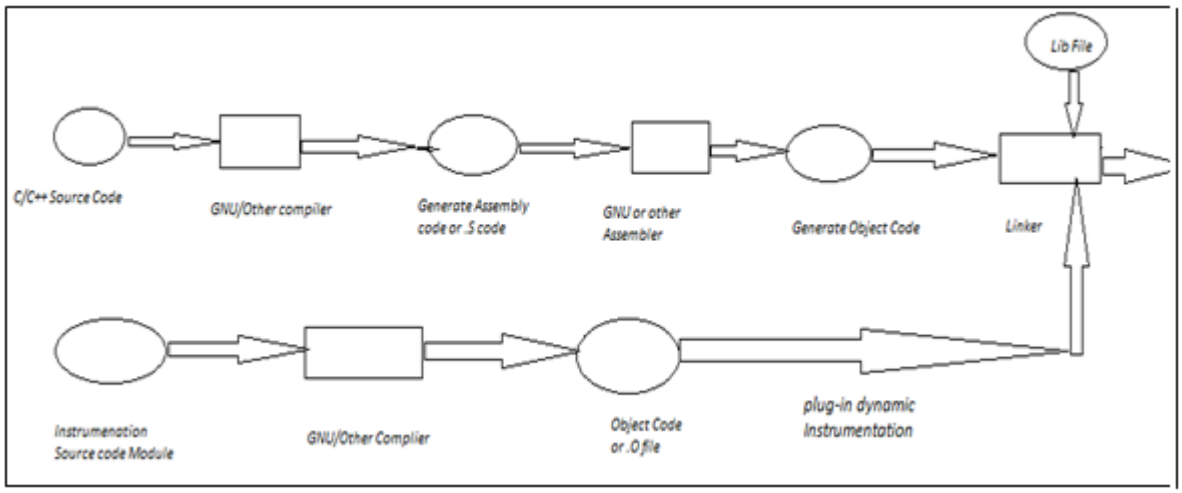

Figure 1:

either statically or dynamically .Instrumentation [7] techniques already shown various avenues in the study of runtime program analysis ,verification and debugging.

3. Existing System and Motivation of Our Work

In most of the existing system the Business Rules can be extract/add only during the inert time period .Further, very few researches actually adapted dynamic binary instrumentation for doing the same. So the main aim of this paper to provide Business Rules that can be add at Run Time through dynamic binary plug-in.

4. Our Approach

We proposed both conceptual and empirical level approach for effectively insert new enterprise based business logics through a separate instrumentation program executable into the existing runtime software system without altering the execution trace of existing system. We divide this section into two parts. In part five, we discuss the conceptual framework of our approach and in part six, we proposed how and where the actual new business rules plug-in will be insert so that new plug-in rules will be integrate with the existing rules at runtime. We take a C-code based example to validate our approach on GNU gcc compiler for windows.

5. Conceptual Approach

Legacy systems requires many aspects of business decisions to be enforce at different time with changing scenario of enterprises business policies. But redesigning such system is complex and costly. Hence, the only promising way to do so is reverse engineering [12, 13] based runtime analysis of such software system and addition of separate logic module(s) with existing runtime executable of the system to make them cope with present industry demands. We propose a framework as shown in figure1 which will integrate the complied executable of newly created business logic module with the existing executable software system through plug-in instrumentation. (Ref. Fig.1)

The existing information system and the new business rule based module must be complied and assembled separately by our propose framework to generate separate ‘object files’. Then all such object files along with library files are passes as a parameter to linker which will generate the compiled and integrate executables as we claim above.

When a program comprises multiple object files like above figure, the linker combines these files into a unified executable program, resolving the symbols as it goes along. The linker also takes care of arranging the objects in a program's address space. This may involve *relocating* code that assumes a specific base address to another base. Since a

Sample legacy program(electricity.c)

```
//project on the billing system for the
electricity company
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include "instru.h"
void main()
{
int customer_no;
char catagory;
float unit1,unit2,unit;
// clrscr();
do
{
printf("\n enter the following letters for
the catagory of the customer");
printf("\n i for INDUSTRIAL");
printf("\n b for BULK INDUSTRIAL");
printf("\n d for DOMESTIC");
printf("\n e for EXIT");
printf("\n enter the customer number:");
scanf("%d",&customer_no);
printf("\n enter the catagory of the
customer");
scanf("%c",&catagory);
printf("\n Enter the previous  and present
meter reading number:");
scanf("%f%f",&unit1,&unit2);
unit=unit2-unit1;
switch(catagory)
        {
        case 'i':

    printf("\n your bill amount is Rs.
%f",unit*10.0);
        break;
        case 'b':
      printf("\n your bill amount is Rs.
%f",unit*12.0);
        break
        case 'd':
      printf("\n your bill amount is Rs.
%f",unit*8.0);
        break;

        case 'e':
           printf("\n Thank You for using
Electricity module");

        exit(0);
        break;

        default:
        printf("you have entered wrong
choice");
   }
}while(1);

getch();
}
```

Our Instrumentation program(Instru.c)

```
#include<stdio.h>
#include<conio.h>
#include "instru.h"
   void auto_generate_bill(int custno,float
smeter, float emeter)
   {

      printf("\n your bill is Rs",(emeter-
smeter)*100);


   }
void dispatch_bill()

{
 Int mobileno;

Printf("\n please register mobile for bill");
scanf("%d",&mobileno);

// insert web based code for sending sms of
electricity bill at &mobileno
}
```

Proposed Dynamic Binary Instrumentation

Auto_generate bill()

Dispatch bill()

Our instru.h(header file)

```
#ifndef INSTRU_H_INCLUDED

#define INSTRU_H_INCLUDED

void auto_generate_bill(int custno,float
smeter,float emeter);
void dispatch_bill();

#endif // INSTRU_H_INCLUDED
```

Figure 2

compiler seldom knows where an object will reside, it often assumes a fixed base location (for example, zero). Relocating machine code may involve re-targeting of absolute jumps, loads and stores which we will try to focus on next part. We prefer dynamic rather than static linking of such object files .A loader output directly to memory is called the *loader*, though loading is typically considered a separate process. Both static as well as dynamic loading can be used under such circumstances.

The plug-in instrumentation helps to monitor or measure the level of a product's performance, to diagnose bugs, add new features at runtime and to write trace information [8].Instrumentation are mainly of two type (source code & binary instrumentation).We choose in this paper dynamic binary instrumentation [9, 10, 11] for obvious reason because they are useful in behaviour analysis of studied system in those situations where source code might not be available. Every DBI framework has a number of important characteristics like ease of tool-writing, robustness, instrumentation capabilities, and performance. There are several advantages of such dynamic binary instrumentation which we are listing below:-

a. No need to recompile and re-link
b. Able to discover code at runtime
c. Able to handle dynamically generated code
d. Can attach to running process
e. Actual behavior of the Program being instrumented remain unchanged
f. Can able to insert, modify features at runtime in system being instrumented

6. Empirical/Experimental Approach

As discussed above, our main aim in this paper is to add new logic modules through runtime plug-in. This shall be a part of legacy software modernization and can be done effectively by addition separate program at runtime to our legacy program by instrumentation. We have taken a demo example of the Electric supply corporation's information system of any city (Ref.Fig.2). In those electricity board legacy system, provision for adding new business rules(like customer cognitive behavioural aspects, automated monthly bill generation and dispatch to customers via emails or sms, enterprise promotion offers)are not very easy as the software evolves over time and integrate new business logics with existing logics are also very difficult. Hence, we propose a binary instrumentation techniques at low level through which original electricity program and instrumentation program (which contains new rules implementation details and bill automation) can be integrate together at execution time without disturbing company's operational tasks.(Ref.Fig.3)

Step 1: The original legacy software program called "electricity.c" and our proposed instrumentation program called "instru.c" are separately assembled first to get assembly files "electricity.s" and instru.s" filed obtained.

Step 2: In this step the assembled file "electricity.s" and "instru.s" are complied separately using gcc complier from window 7 to get the object file " electricity.o" and "instru.o".

Step 3: In this phase both the output files "electricity.o" and "instru.o" of step 2 are commonplace to combine together as a single object file "electricity_instru.o" using Linker. The compiler produces an intermediary form called object code. Object code is often the same as or similar to a computer's machine language. Object code is a portion of machine code that hasn't yet been linked into a complete program. It's the machine code for one particular library or module that will make up the completed product. It may also contain placeholders or offsets not found in the machine code of a completed program that the linker will use to connect everything together.

Step 4: This is the last phase of our prototype where step 3 output file is converted into executable file "electricity_instru.exe" which runtime will integrate the existing business logic of electricity Supply Corporation with the newly implemented logic using plug-in. Major consideration:

- As C is not binary standardized language hence for different compiler or for different versions or settings on the same compiler , the DLL will be generated differently and may cause crashes with the application it is linked to.
- It should be noted that combining multiple object files together as a single object file is not a easier task When linking libraries must to be ordered and can't handle cyclic dependencies.
- Instrumentation itself has certain issues which needed to be thoroughly analyzed before actual real-life application.

7. Result & Discussion

As mentioned earlier, we takes two separate programs "electricity.c" and "instru.c" which are assuming as an original electricity billing program and instrumentation program to add new business modules respectively. We separately assembled and compiled both of these file using gcc for windows. Finally, we attempt to combined both the object files "electricity.o" and "instru.o" as a single shared object file called "electricity_instru.o" using the steps mentioned above. For simplicity, we kept only one customize function in "instru.c" file called void dispatch_bill(int customerid) function. This function takes the customerid as parameter and dispatch bill to the mobile no. that customer will entered at runtime. The final snapshot of the output is given in Fig.4 with arbitrary inputs.

8. Conclusion & Future Work

This dynamic approach is actually an attempt towards reverse engineering and provides an idea of binary instrumentation implementations. The final module "electricity_instru.exe" may be able to integrate both the features of differently object files one of which i.e. the actual legacy program of our case study and another program contains the newly required business criteria's which organization demands to integrate with their existing

```
C:\Users\om\Desktop>gcc -S electricity.c   [Convert electricity.c to electricity.s assembly file]

C:\Users\om\Desktop>gcc -S instru.c        [Convert instru.c to instru.s assembly file]
```

```
C:\Users\om\Desktop>gcc -c electricity.s [Convert electricity.s to electricity.o object file]

C:\Users\om\Desktop>gcc -c instru.s       [Convert instru.s to instru.o  object file]
```

```
C:\Users\om\Desktop>gcc electricity.o instru.o -o electricity_instru.o
```

```
C:\Users\om\Desktop>gcc electricity_instru.o -o electricity_instru.exe
```

Figure 3

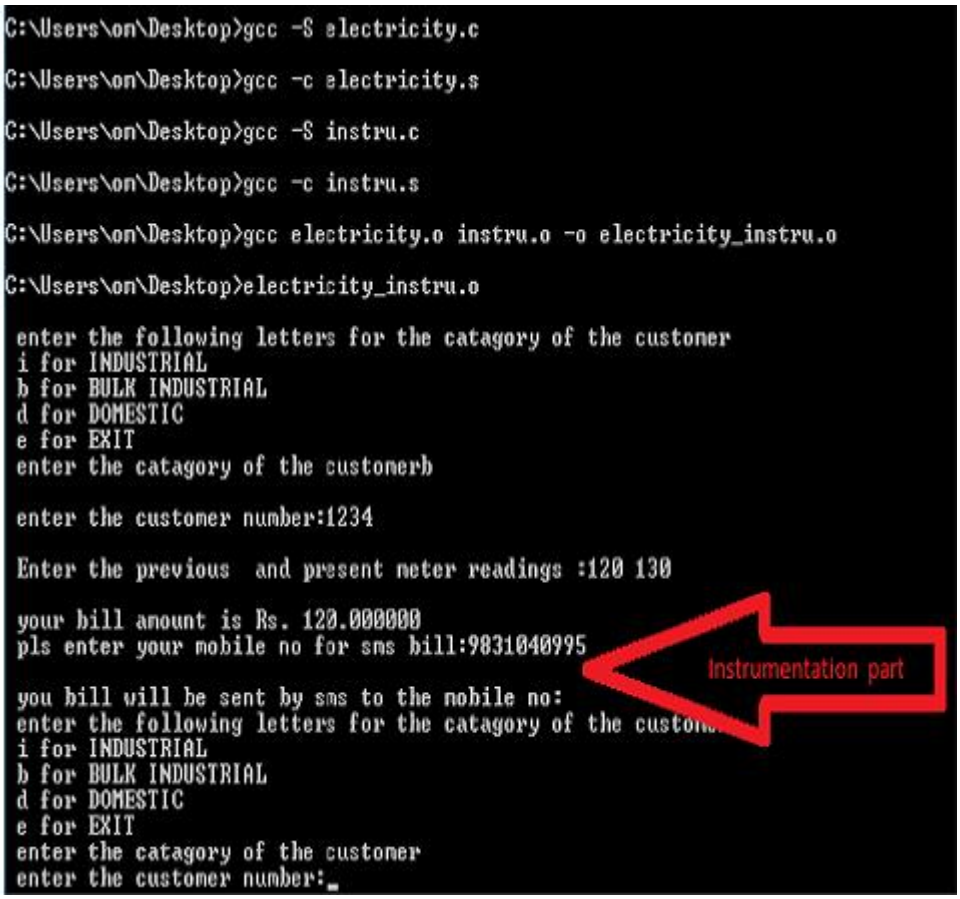

Figure 4

information system without disturbing routine business operations. However, there are many issues arise with such Plug-in [14] approach we mentioned which need to more effectively answered in recent future before such instrumentations may be implemented in real life situation for legacy modernization.

References

[1] O. Vasilecas, "Ensuring Consistency of Information Systems Rules Models", Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, Rousse, Bulgaria, June 18-19, 2009 .

[2] A. Charfi, "Hybrid Web Service Composition: Business Processes Meet Business Rules". Proceedings of the

2nd international conference on Service oriented computing, ACM, New York, USA 2004, pp 30-38.

[3] N. Zsifkov, "Business Rules Domains and Business Rules Modelling". Proceedings of the International Symposium on Information and Communication Technologies, Las Vegas, Nevada, USA, June 16-18, 2004, pp 172 – 177.

[4] E. Putrycz and A. W. Kark, "Connecting legacy code, business rules and documentation" , Proceedings of the International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008, pp. 17–30.

[5] T. Systä, P. Yu, H. Müller, "Analyzing Java Software by Combining Metrics & program visualization", Proceedings of the Conference on Software Maintenance and Reengineering ,2000, IEEE Computer Society Washington ,USA, pp 199.

[6] T. Kempf, K. Karuri, L.Gao "Software Instrumentation", Wiley Encyclopedia of Computer Science and Engineering, published online. New Jersey, 15 September, 2008.

[7] B. R. Buck and J. Hollingsworth, " An API for runtime code patching" , Journal of High performance Computing Applications, 14(4): pp 317-329, 2000.

[8] Source Code Instrumentation Overview at IBM website. Available: http://www-01.ibm.com/support/knowledgecenter/SSSHUF_8.0.0/com.ibm.rational.testrt.doc/topics/cinstruovw.html . [Accessed: Jan. 12, 2015].

[9] C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S.Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation", Proceedings of PLDI 2005, pages 191–200, Chicago, Illinois, USA, June 2005.

[10] J. Maebe, M. Ronsse, and K. De Bosschere, "DIOTA: Dynamic instrumentation, optimization and transformation of applications", Proceedings of WBT-2002, Charlottesville, Virginia, USA, September 2002.

[11] N. Nethercote and J. Seward, " Valgrind: a framework for heavyweight dynamic binary instrumentation" , Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation ,pages -89-100,ACM New York, USA.

[12] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: a generic and extensible framework for model driven reverse engineering," in : Proceedings of the IEEE/ACM Internaltional Conference on Automated Software Engineering, ASE 2010, pp.173-174.ACM ,New York 2010

[13] F. Barbier, G. Deltombe, P. O., and K. Youbi, "Model Driven Reverse Engineering: Increasing Legacy Technology Independence" , Proceedings of Workshop on Reverse Engineering , Feb 23 - 24, Thiruvananthapuram, India 2011.

[14] D. Das and P. Kundu, "An Attempt to Analyze & Resolve the Pitfalls in CRM Software through Plug-In Instrumentation", *International Journal of Scientific and Research Publications* [ISSN 2250-3153], Vol. 2, Issue 5, pp. 313-320, May 2012.

Author Profile

Prasenjit Kundu is currently pursuing doctoral degree program in computer science in Utkal University, India

Dr. Bikram Kesari Ratha is currently the Reader in the department of computer science & application in Utkal University, India