Pattern Language and Traditional Programming Practices for Exporting Functionality

Dr. Hari Ramakrishna, Professor,
Department of Computer Science and Engineering, C.B.I.T Gandepet , Hyderabad- 500075
Email: dr.hariramakrishna@rediffmail.com

***Abstract*— A set of programming practices and a model pattern approach to document and communicate expert programming tips are presented. Pattern approach is adapted to the task of exporting functionality. Traditional development frameworks which are presented in integrated development environment are adopted to suit the implementation of the presented models. Extreme Programming (XP) approach for change management is adopted. The model starts its journey from known simple techniques to complex models. Such models are useful for training programmers. The presented models depend on concepts such as static function , polymorphism, virtual functions, static and dynamic libraries, Object files, managing project work spaces, code COM models, ATL and Active X controls , Simple Object access protocol SOAP. This paper presents a new approach to train the programmers.**

***Index Terms*— Pattern, Pattern-frames, Frameworks, Pattern language, Extreme programming (XP), Refactoring, Object oriented primitives, Component technology, Function classes, Middleware frameworks**

I. INTRODUCTION

Software industry is looking for rapid application development mechanisms with client orientation and short time span delivery, increasing quality and withstanding rapid changes in technology and requirements.

In this connection, exporting third party tools plays a major role. Using third party tools decreases testing time. Development of frameworks for increasing degree of reuse has become an important focus. Customization of such frameworks as per client requirements increases importance of frameworks. Frameworks are different from libraries; client code is embedded in frameworks whereas client code includes and calls libraries.

Frameworks can be classified into three classes namely 'system frameworks', 'middleware integration frameworks' and 'business oriented frameworks'. System frameworks provide basic reusable environment required for the development, middleware integration frameworks provide Integrated Development Environment (IDS); for example, Microsoft MFC, Document view architecture, COM Framework, ATL, .NET frameworks come under such types. The third class of frameworks focuses on building and exporting business functionality focused on one or more domains. Such frameworks help in increasing quality and decreasing development cost and time.

Exporting data and functionality, and integrating and assembling modules for building applications are important tasks in such models. Several programming practices and styles, useful for such purpose, are presented as a series of techniques, in the increasing order of complexity starting from basic known concepts. Extreme Programming (XP) and refactoring also suggest such practices.

Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. It is an agile software process, it advocates frequent "releases" in short development cycles. The approach is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted. Such approaches enable programmers to withstand and adopt multidimensional rapid growth and changes in the software development technologies.

The presented models which are experimented in several applications are adopting such processes. They focus on presenting various techniques in practice for exporting functionality for the purpose of building function libraries and frameworks, starting from traditional basic concepts of refining code and adopting object oriented concepts. At the end, a new concept referred as 'function classes, is presented. The paper concludes with techniques for porting such models into upcoming technologies in a simple and safe way. Pattern approach of documentation and communication of professional practices is advised for such purposes.

II. PATTERN APPROACH FOR DOCUMENTATION OF PROFESSIONAL PRACTICES

Pattern approach for documenting and training expert skills was introduced by Christopher Alexander and was applied in architectural domain [4]. He has proposed a pattern language consisting of a series of patterns for improving quality of architectural designs. The same was suggested by Grady Brooch to Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides who are well-known by Group of four (GOF). They found a set of solutions for problems in design issues which are known as design patterns, which are classified as creational, structural and behavioral patterns [6]. Microsoft COM Technology uses several design patterns for building COM frameworks. Programming in Component Technology needs understanding of these design patterns.

GOV known as Group of Five presented Architectural patterns. Frameworks are defined as semi completed reusable applications, using patterns for building

DOI:10.32377/cvrjst0501

applications in an efficient way. Frameworks are domain specific unlike design patterns.

A series of patterns which address a common problem are named as pattern language. Pattern language is useful for documenting expert skills for communicating and training professionals. Coplien presented C++ programming styles as a set of patterns. The pattern language mechanism is in use for presenting Java-based programming models. Coplien says that a good pattern solves a problem, is a proven concept, describes relationships, contains a significant human component such as comfort, quality of life etc.

Several pragramming practices have been in use before evolution of the pattern concepts. As they are very common and well known they will be generally referred to as primitive patterns. For example, Object Oriented Programming addresses several programming issues and presents several concepts and environment implementation of these concepts. Programming practices and concepts such as Polymorphism, function overload, static functions and data, exception handling, inheritance, interface, abstraction, encapsulation, friend functions, inline functions and several object oriented features are reffered to as primitive pattern.

The main reason to call existing programming features like object oriented features as primitive patterns is to use them as participating in forming a pattern language. Each primitive practice has a purpose and design to solve a set of problems. Giving pattern light to programming style makes programmer understand the specified purpose of usage of these concepts.

III. A JOURNEY TO OBJECT ORIENTATION

This section presents the need of object orientation in the light of managing functionality in the Extreme Programming way of development in small cycles, refactoring for client requirements and change management (from simple C to C++). A simple first step programming in C for handling requirements of a list of elements represented as a matrix is considered as base requirement as presented in Table 1.

The purpose of the example C program is to read, store, sort and print a list of elements of user choice. Finding the limitations and reconsidering the requirements is referred to as refactoring (in XP), and the process helps in evolving and introducing new concepts. By repeating such process leads to the evolution of new programming methodologies and models.

TABLE I.
C PROGRAM FOR MANAGING MATRIX OPERATIONS

```
void main()
{
<<   1. code for   Declare data  variables >>
            int List[10];
            int  Count, totalNoOfElements, temploc;
< <  2. Code for Initialize variables  >>
<<   3.  Code for Read the list of  Elements  >>
<<   4.  Code for Print the Input  List >>
<<   5. Code for Perform Sorting operation >>
<<   6. Code for Printing  Sorted List >>
<<   7. Code segment for Exiting  Application >>
}
```

For example, generalizing concepts of given C program in the light of reuse, the new concept, namely template, evolved. Templates are created in several languages to store and sort any element types to reuse the procedures. Further generalization of concepts leads to evolution of design pattern 'Template'.

Template is a design pattern (behavioral pattern). As per GOF, *Template is intended to define the skeleton of an algorithm in an operation, deferring some steps to subclasses*. The Template method lets subclasses define certain steps of an algorithm without changing the algorithms structure.

TABLE II.
A PROGRAM IN C FOR MANAGING A SIMPLE LIST OPERATIONS

```
typedef struct
{
        int List[101];
        int m_itotalnos;
} DataList;
<< Define function to perform operation on data  >>
void ReadDataList(DataList* Data)
void DisplayDataList(DataList data)
void SortDataList(DataList* Data)
void AddToDataList(DataList* Data,int Ele)
void Exit()
<<  Write main  program which is a client for these above reusable code
segments >>
void main( )
{
   DataList MyData; // Declare List
   ReadDataList(&MyData); //  Read List
   printf("The input list is shown below \n "); // Print List
   DisplayDataList(MyData); // Sort DataList
  SortDataList(&MyData); // Print Sorted List
   DisplayDataList(MyData); // Print List
   Exit();// Exit program
}
```

As a next step to the selected example, the following modifications improve the quality of the program. The new program structure with new changes is presented in the Table 2.

i) Introducing functions, code segment are reused

ii) Defining a proper data type enables data binding problems to be handled.

DOI:10.32377/cvrjst0501

Refactoring the changed program creates scope for new client requirements.

i) Requirements related to packing data and procedures defined on it forming a single entity
ii) Techniques for exporting the functionality
iii) Addressing Encapsulated issues
iv) Requirements related to configuration of functionality
v) Exporting functionality across the process, in a distributed environment and over the internet.

These questions raise the need to use object oriented features. A simple sample class in C++ is presented in table: 3.

TABLE III
A CLASS IN C++ FOR MANAGING FOR SIMPLE LIST OPERATIONS

```
class DataList
{
private:
        int m_ List[101];
        int m_iTotalNos;
public:
        DataList(void); // Constructor
        void ReadDataList();
        void DisplayDataList();
        void SortDataList();
        void AddDataList();
        void AddToDataList(int Ele);
       ~DataList(void); // distracter
}
```

Typical practices and techniques related to exporting functionality are addressed in the following sections.

Iv. EXPORTING FUNCTIONALITY THROUGH LIBRARIES

The project work space concepts and creating a directory structure are essential for exporting functionality. Microsoft Integrated development environments (IDE) provide several frameworks for managing project work spaces suiting several requirements. The management of directory structure provides good program code management practices; it is also helpful for sharing files across different project workspaces. Sample generic directories used frequently are BIN, DEGUD, RELEASE, SRC, INCLUDE, OBJ, LIB and DEF. Creating proper directory structure is an essential feature of complex projects where several places of project development share files across projects. Sometimes the directories are shared across Internet using FTP protocols. Visual sources like code management tools also suggest proper directory structure for code management where several programmers work in parallel and share common files for development and bug fixing.

The *static data* and *static function* are known as class members. But the token word static represents limited scope, which means a static function defined in a source file exists and extends its visibility to that file scope only. This behavior of static functions can be used for exporting functions across files within a project workspace encapsulating the internal implementation and dependent functions. Object files carry the implementation code and headed files carry interface function definitions which are exported to the client. This procedure of exporting functions through object files is presented in Table 4:

TABLE IV.
EXPORTING FUNCTIONS THROUGH OBJECT FILES

SERVER PROCEDURE:

i) Define internal functions which have no relevance to client and which are required to perform client task through interface functions as static functions.
ii) Define interface functions which are exported to the client as non-static functions
iii) Define prototype of these in a header file and keep the header in a INCLUDE directory.
iv) Compile the source file which generates an OBJ extension object file; this can be directed to OBJ directory with suitable settings at project workspace.

CLIENT PROCEDURE:

Include the header file in client code and attach the object to project workspace.
Use the interface functions defined in header file at client code irrespective of hidden dependent functions.

The next model presents exporting through static and dynamic libraries using special project files. IDE of Microsoft provides projects workspace for static and dynamic libraries separately. The static libraries attach functions to executables at the time of linking. This will increase the size of executables in proportion to included functionality.

In dynamic libraries, the functionality is attached through a dll extension file at run time. In dynamic link libraries the executable file needs to carry all the dependent dlls. The lib file generated at the time of building the server application carries the dll path and other required information.

The dynamic library concept makes the executable files lightweight. Large scale applications can be loaded with limited RAM as the executable size is very low. In huge applications, the total functionality supported and implemented in an application is larger than the functionality required at a particular time of execution for performing task of that time. The operating system automatically unloads the unused dlls and loads required dlls dynamically for optimizing the internal memory.

This model of exporting functionality has another advantage; it allows loading required (purchased or permitted) functionality alone to be distributed to clients. In case the required dll file is not available during run time, due to some problem (or unauthorized usage), the operating system will raise an exception and give a message to the user, thus preventing runtime error problems.

Implementations of these models need a '*definition file'* with def extension. A sample def file is presented in the Table 5. The functions defined in def file alone will be exported to the client project. The project space will

DOI:10.32377/cvrjst0501

automatically encapsulate the internal details. Defining internal functions as static functions is not required in this model.

TABLE V.
A MODEL SAMPLE DEFINITION FILE FOR BUILDING A DYNAMIC LINKED LIBRARY SCRIPT FOR THE SRC/Srever.def FILE

```
LIBRARY         DLLSERVER
;CODE           PRELOAD MOVEABLE DISCARDABLE
;DATA           PRELOAD MOVEABLE SINGLE
HEAPSIZE        1024
EXPORTS

   InterfaceFunction01          @1
   InterfaceFunction02          @2
```

In case of function overload (same name is used for more than one function with different arguments), the name of the functions along with arguments (generated by the compiler) which are available in the library and object file will need to be used in definition file.

V. FUNCTION CLASSES

This section presents framework model for exporting functionality. This is referred to as '*function classes'*. The function class is a class which has only functions or methods defined without data like interfaces. Interface classes are abstract but function classes are not abstract classes; they include implementation. In general, a class has data (to compute the state of object) and methods (to implement behavior) that work on data to change object state. The function class is coined only to export functions with client-oriented features using object primitives.

In applications like CAD, GIS we find lot of function libraries at different levels. For managing and exporting functions in a professional way function classes provide solutions. Several object oriented features (primitive patterns) can be applied on these classes to make these libraries user friendly and allow client to configure the inherent procedures as per requirements in an authorized way. For example if client want to define his own model which is used in building the interface function client can do such operations using object oriented features for managing multiple behavior.

Table 7 will present sample code segment required to export a function class. In the given sample macros are defined to manage a uniform header file for class definition which will present the definition as per the requirements of client and server. A brief description of function class is presented in Table: 6

TABLE VI
THE FUNCTION CLASS PATTER-FRAME

Name: Function-class pattern-frame

Intent: The intent of this framework is to manage scalable function libraries.

Motivation and Applicability: Several scalable function-groups are required in the domain of graphic, CAD and GIS. Such requirements are managed using such pattern-frame models.

Structure: The Architecture of function–class frameworks is presented in Figure 1.

Participants: Abstract Function class: This class defines the function – groups.

Function class: This class implements the function groups

Abstract Primitive: This class is used for defining the configurable function-groups.

Primitive Library: This class is used for implementation of configurable function-groups

Collaboration and consequences: The Function class implements the abstract function class which is an interface. This implementation depends on Abstract Primitive which is another interface. The primitive library which is supposed to implement the abstract primitives is used for configuring the function-groups. The client can add his own primitive library for configuring the function group as per the requirements. Sample Code segment for implementation is presented in table 8.

TABLE VII
A MODEL CLASS DEFINITION FOR EXPORTING A CLASS

```
#ifdef BEEPSERVER
class __declspec( dllexport ) CBeep
#else
class __declspec( dllimport ) CBeep
#endif
{
public:
void Interface Function01(void);
void Interface Function01(void);
};
```

TABLE VIII
SAMPLE CODE SEGMENT FOR FUNCTION CLASS

```
class AbstractPrimitive
{
public:
 virtual   Line (<<list of argument >>) = 0;
 virtual  EllipticalArc (<<list of argument >>) = 0;
}
class AbstractGraphicFunctions
{
public:
virual   Rectangle(<<list of argument >>)) = 0 ;
virual   Square(<<list of argument >>)) = 0;
virual   Polygon(<<list of argument >>)) = 0;
virual   Circle(<<list of argument >>)) = 0;
virual   Ellipse(<<list of argument >>)) = 0;
};Class GraphicFunction : public AbstractGraphicFunctions,
AbstractPrimitives
{
   << Implements Abstract functions and depends on AbstractPrimitives
>>
};class PrimitiveLibary: public AbstractPrimitives,GraphicFunctions
{<<  Implementation of AbstractPrimitives >>
}
```


DOI:10.32377/cvrjst0501

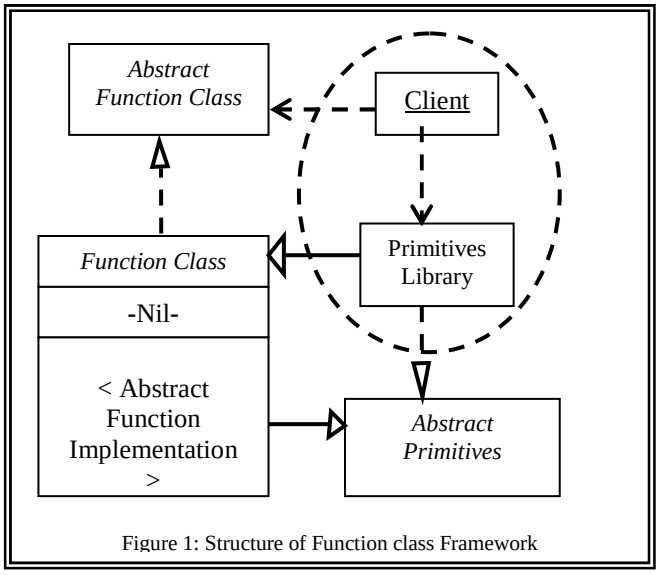

Figure 1: Structure of Function class Framework

The above classes do not have any data members. They have only functions. The Architecture of this function class framework is named as function class pattern-frame. This is different from the set of function libraries. Important features of function class framework as compared with a function library are listed below.

i) The Function class frameworks allow user to configure to the user requirements. As an example, the user of the above class wants to use the above class, but he wants to use another DDA algorithm instead of the simple DDA algorithm both for line and elliptical arc. For achieving this, client will inherit from the abstract-primitive class and will implement the virtual functions.

ii) The client can extend the function class by adding some more graphic primitive procedures. In this case, the client defines another abstract function class. The abstract-function class of the existing framework will become abstract primitive for the new extended framework. This procedure can be used in an iterative way.

iii) Exporting a class is more efficient than exporting a set of functions.

iv) These function classes are not objects and they do not have state. They exhibit behavior. They can be treated as a package of functions.

v) It is possible to implement these function classes even in other languages like Java. Any language that implements Object oriented patterns such as inheritance and polymorphism, and supports Dynamic linked libraries, is suitable for implementing function classes.

vi) The Dynamic Linked Libraries make the function classes loosely coupled with the Applications. This will enable changing the implementation of function classes without affecting the client application or modules, thereby making function-classes scalable.

V. MIDDLE WARE FRAMEWORKS FOR CREATING COMPONENTS

The middle ware frameworks provided in IDE allow exporting functionality in modern and other complex technical environment in a simple way through a set of wizards. The COM DLL enables a procedure to export functionality over COM interfaces which are more efficient in management but core COM object implementation is complex. Several frameworks like Active Template Library for creating ATL objects are available in IDE for exporting over COM interfaces. Several pattern-frames are designed to handle COM model simple to use [3]. The SOAP simple object access protocol is also providing frameworks for exporting functionality as web services. All these frameworks work like wrappers over existing traditional models of exporting. Figure 2 presents a middleware integration framework for exporting object oriented frameworks using IDE and middleware interaction frameworks for components. The same can be adopted for other IDE models.

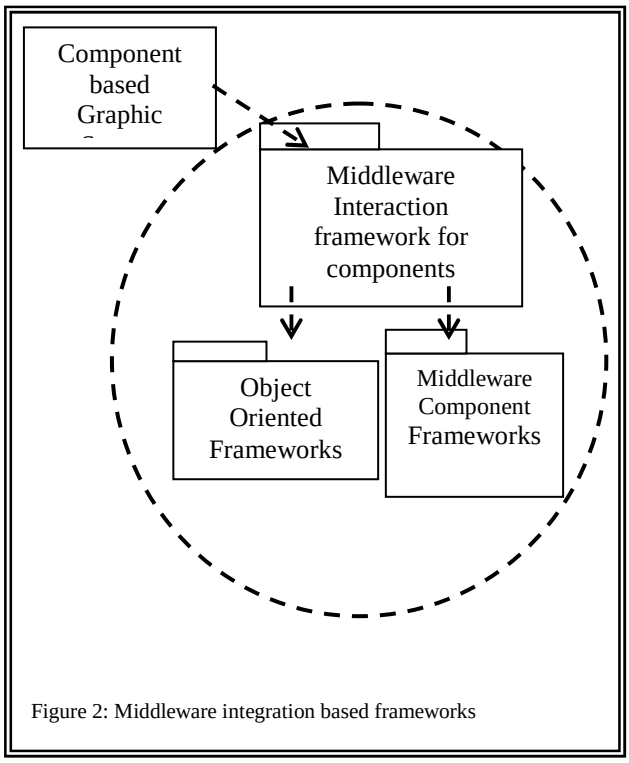

Figure 2: Middleware integration based frameworks

DOI:10.32377/cvrjst0501

CONCLUSIONS

Various programming practices for exporting functionality are presented starting from simple unstructured first level programming is C. The method of refactoring for changing to new model as specified in extreme programming XP is adopted along with a pattern way of presentation. The former presented models referred to as programming tips are styles and latter are known as pattern-frames. The later models use object oriented pattern primitives to solve the problems. Presented pattern frame exports functionality in a user friendly way, enabling client applications to configure the functionality as per requirement. These models can be further exported to new technologies using frameworks available in IDE. Some of the pattern frames are not discusses in detail as they are beyond the scope of this paper. The solutions are aimed at training programmers for efficient programming related to a particular problem. Such sequence of models for domain specific problems forming a pattern language is suggested.

ACKNOWLEDGMENT

The author wishes to thank Dr. K.V.Chalapati Rao, Dean-Research, CVR College of Engineering and also the faculty of the Department of Computer Science and Engineering C.B.I.T, for their support and encouragement.

REFERENCES

[1] Dr.Hari Ramakrishna, Dr. K.V Chalapati Rao,"Pattern Methodology of Documenting and Communicating Domain Specific Knowledge", CVR Journal of Science & Technology , Volume 2, June 2012 ISSN 2277-3916.

[2] Dr.Hari Ramakrishna, "Pattern Approach to Build Traditional Graphic Frame works", International Journal of Computer Applications Volume 59– No.15, p35-42, December 2012. Published by Foundation of Computer Science ISSN :(0975 – 8887), New York, USA.

[3] Dr. Hari Ramakrishna, "Design Pattern for Graphic/CAD Frameworks", Ph.D thesis submitted to Faculty of Engineering Osmania University March 2003.

[4] Christopher Alexander, "An Introduction for Objectoriented Design", A lecture Note at Alexander Personal web site www.patternlanguage.com.

[5] Pattern Languages of Program Design. Edited by James O. Coplien and Douglas C. Schmidt. Addison-Wesley, 1995.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns: Elements of Reusable Software Architecture", Addison-Wesley, 1995.

[7] LNCS Transactions on Pattern Languages of Programming http://www.springer.com/computer/lncs?SGWID=0-164-2-470309-0

[8] Hari RamaKrishna "COM Applications for Real time Electrical Engineering Applications" IEEE sponsored International Conference at Bangalore - 2000.

[9] Hari RamaKrishna "COM as new Object Oriented Technology", Proceedings of CSI conference, December 1999 at Visakhapatnam.

[10] Hari RamaKrishna, "Application of computer graphics in interior design" Proceedings of Conference 1998 at Institutes of Engineers at Hyderabad.

[11] Hari RamaKrishna "Generation of flooring and wallpaper patterns using computer graphics" Proceedings of the First National Conference on Computer Aided Structural Analysis and Design, Jan 3-5,1996, Engineering Staff College of India and University College of Engineering, Osmania University, Hyderabad.

AUTHOR'S PROFILE

Dr. Hari Ramakrishna was awarded B.E in Computer Science and Engineering in 1989 by Osmania University, Hyderabad, A.P., INDIA, M.S., in Computer Science by BITS PILANI, INDIA and Ph.D. in Computer Science and Engineering by the Faculty of Engineering Osmania University in "Pattern languages for graphic /CAD frameworks". He has worked in Software Industry for several years developing Graphic, CAD /GIS products using Microsoft environment. He has about 16 years of teaching experience. Presently he is working as a Professor for last 8 years in the Department of Computer Science and Engineering at Chaitanya Bharathi Institute of Technology, Hyderabad INDIA. He is involved in the design and development of several graphic frameworks for various Engineering applications

DOI:10.32377/cvrjst0501