An Application Programming Interface For Developing Distributed Algorithm Along With Proposed Meta Language Concept

Kishalay Bairagi

Abstract: "In computer science, an application programming interface (API) is an interface that defines the ways by which an application programming may request services from libraries."[7] Libraries of a programming language are the list of all classes and interfaces, along with their fields, constructors and methods that are part of that language. For example, java is an object oriented programming language which has a rich set of built-in classes and interfaces packaged in the API also known as java API[7]. Therefore, a programmer can easily borrow built-in classes or interfaces to use the fields, constructors and methods of those classes and interfaces in his or her application and is able to be free from the hazards of thinking the implementation details of those functions and constructors and writing it down to the application he or she is developing. An API [7] also helps a programmer to write a short and compact code, to save time of program and application development and to produce a quality code having readability and understandability than the code without having the application of API. Almost all the modern programming languages come up with the rich set of APIs. The basic difference between an API and library lies in the fact that while "API reflects the expected behaviour, library is an actual implementation of this set of rules". [7] On the other hand, "the relation with framework is based on several libraries implementing several APIs but instead of normal use of an API, the access to the behaviour built into the framework is made possible by extending its contents with new classes and interfaces".[7] This paper presents a component of framework[4] where the API for the distributed algorithms has been plugged into the framework so that a programmer can get services from the built- in classes and interfaces for easily understandable, compact and faster program development. Here a concept of meta language consisting of very simple constructs has been introduced in order to make the language user friendly.

Index Terms: Application Programming Interface (API), Broadcasting, Compactness, Distributed Algorithms, Distributed System, Efficiency, Framework, Library, Meta language, Multicasting, Nodes, Occam, Package, Readability, Understandability, Unicasting.

1 INTRODUCTION

THIS is a known fact that independent nodes collectively build a distributed system [5] and such a system is characterized by its built in uncertainty. Also it is very difficult to write programs for a distributed system because of the demands for heterogeneity, synchronisation [2], consistency, fault tolerance etc. Writing a separate code for each independent participating node is also a tedious job. This paper presents a framework that provides many functionalities of which only one aspect of these functionalities has been discussed. The framework has been provided with an API which has not only removed many intricacies of writing programs for distributed system but has also helped to write compact and easily understandable code. Programmers are now able to write code for distributed system at a rapid rate. The API for the framework has mainly covered the communication aspect of the distributed system. During the execution of the algorithm different nodes in the distributed system communicate and exchange messages to one other. The communications among nodes are of three types i.e. they are unicasting, multicasting and broadcasting. The framework provides the necessary API where the classes and interfaces necessary for these three kinds of communications are available. During the specification of communications among the nodes in the distributed system a programmer only needs to call the methods of sending and receiving messages instead of writing the implementation details of those methods. Programmers can use the services of these methods by importing the packages [1] at the start of the program where the classes for unicasting, multicasting and broadcasting are residing. In this framework Bully Leader Election algorithm [9] has been taken as a reference algorithm and it has been possible to develop the code of the algorithm for two nodes in the distributed system. The algorithm has been tested for three nodes also. The framework has the provision for developing programs of other standard distributed algorithms as well as of user defined algorithms using the API of the framework and using any number of nodes.

- *Kishalay Bairagi is M Tech in computer science and engineering in West Bengal University of Technology, India, PH-9434506175.*
- *E-mail: kishalayb29@gmail.com*

2 Implementation Details of Bully Leader Election Algorithm Using the API of the Framework

This algorithm has been implemented in a distributed system in which participating nodes are connected in mess topology with unicasting. This algorithm has been tested using both two and three nodes. It has been implemented not only in single machine by creating two or three virtual nodes but also in the actual environment i.e. by using two or three computers connected in mess topology. When the framework understands that a request for the execution for BullyLeaderElection algorithm has come it asks the user to enter the pid (process id or node id) of the initiator [9] (who is initiating the program). This is the special input for the algorithm. All possibilities for this algorithm are implemented in the skeleton of the algorithm. Thus the skeleton has three parts- (i) pid=initiator (ii) pid<initiator and (iii) pid>initiator. Each participating node knows its own id i.e. its own process id and the process id of the initiator i.e. the initiator id. At the time of execution of the algorithm participating nodes play their roles according to their pid and mutable initiator. Lamport's Logical time stamping algorithm [3][6] is used to maintain local time stamp [3][6] of the events, which are then connected to a total ordering of events [3][6]. For each node there are two threads

which execute – (i) one is sending thread and the other is receiving thread. The sending thread uses the services from the class UnicastSend which resides in the DATAGRAM package and the receiving thread uses the services from the class UnicastReceive which is also under DATAGRAM package. Both the UnicastSend and UnicastReceive have several constructors and methods which provide different kinds of services according to the needs of the programmer. In order to use the services of these classes, what a programmer needs to do is to import the DATAGRAM package in his or her application. In this way the API helps a programmer to develop application by providing the services from its library and also helps to write more efficient, compact, manageable and understandable code. A small structure of code snippet will illustrate the above facts.

```
package DATAGRAM;

import java.io.*;

import java.net.*;

public class UnicastSend{

DatagramSocket mySocket;

public void send( DatagramSocket mySocket) throws IOException{

// implementation details of the send function goes here
}

public void send( int receiverPort,String message) throws IOException{

// implementation details of this version of send function goes here

}

public void send( InetAddress receiverHost, DatagramSocket mySocket, int receiverPort, String message) throws IOException{

// implementation of this function also goes here

}
```

3 Scope for Future Work

The framework has the provision to accept Meta language which is under development. Meta language is the language which is user friendly. Therefore, a programmer will have to write fewer lines of code to develop distributed algorithm. The Meta language can be tag based like html mark up language or it can also be written in the form of language which consists of very simple construct. Occam [8] is such kind of language. Therefore, Meta language is a language which defines another language. The Meta language precompiler compiles the code written in Meta language and generates the corresponding java code. It is also hoped that in the near future using the API of the framework more number of distributed algorithms can be implemented. Then API will become more general. An example of Occam construct has been given below in order to show the simplicity of the construct. For example, if several processes are to be run concurrently ("true parallelism or simulated parallelism by time slicing" [8]) 'PAR' [8] construct can be used. It is a very simple construct to execute any number of processes parallel. 'The whole construct terminates when all of the PAR's components terminate. Here is the structure of the 'PAR' construct."[8]

```
"PAR
  p1
  p2
  p3"
```

It can easily be understood that here p1, p2 and p3 are three processes which have been made to run concurrently by calling them under the 'PAR' construct. It is to be noted that "the whole construct terminates when all the three p1, p2 and p3 terminate."[8]

4 Conclusion

Every framework comes up with a rich set of libraries, and the API is the way to get the services from these libraries. The framework which has been mentioned here is not an exception. But the speciality of the framework lies in the fact that it provides API for the development of distributed application. Writing code and developing application for the distributed system are not easy tasks. By the virtue of this API, programmers are relieved of many intricacies of writing programs for distributed system and they are able to concentrate on better program logic, compactness, readability, efficiency and performance of those algorithms.

Acknowledgment

The author thanks his parents and wife.

References

[1] http://docs.oracle.com/javase/7/docs/api/java/util/package- summary.html

[2] http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ locks/package-summary.html

[3] Lamport Leslie, "Time, clocks and the ordering of events in a distributed system", Communications of the ACM, v. 21, no. 7, pp. 558-564, July 1978.

[4] NS-3(Statistical Framework) http://www.nsnam.org.

[5] Pheatt Chuck, Department of Computer Science, Emporia State University: ''An Easy to Use Distributed Computing Framework''.

[6] Raynal Michel, Singhal Mukesh, "Logical Time: Capturing Causality in Distributed Systems", IEEE Computer Society Press Los Alamitos, CA, USA Volume 29 Issue 2, February 1996 Page 49-56.

[7] Application Programming interface- Wikipedia, the free encyclopaedia en.wikipedia.org.

[8] Introduction to the Programming Language Occam by Dr. Daniel C. Hyde, Department of Computer Science, Bucknell University, Lewisburg, PA17837, hyde@bucknell.edu.

[9] Silberschatz Abraham, Galvin Peter Baer Galvin, Gagne Greg Gagne, “Distributed Synchronisation”, Operating System Principles, Wiley, John Wiley & Sons. INC.