Multimodal Pattern-Oriented Software Architecture for Self-Configuration and Self-Healing in Autonomic Computing Systems

Vishnuvardhan Mannava*
Department of Computer Science and Engineering
K L University
Vaddeswaram, 522502
Andhra Pradesh, India
vishnumannava@acm.org

T. Ramesh
Department of Computer Science and Engineering
National Institute of Technology
Warangal, 506004
Andhra Pradesh, India
rmesht@nitw.ac.in

ABSTRACT

Because of the diverse nature of software systems, it is unlikely that systems will be developed using a purely service or component programming paradigms. Therefore, the ability to combine the strength of various programming paradigms and use them in a complementary manner becomes essential. As far as we know, there are no studies on composition of design patterns and pattern languages for autonomic computing domain. The work presented in the paper is concerned with composition of existing design patterns which are taken from various programming paradigms that are used for developing of the self-configuration and self-healing characteristics of the autonomic computing systems. In this paper we propose multimodal pattern-oriented software architecture with composition of Worker Object, Look-Up, Row Data Gateway Database access, Adaptation Detector, Case-Based Reasoning, Leader/Followers,and Observer design patterns using Java Web Services (JWS), JUDDI service repository, and Java Aspect Components (JAC) Frame work by Providing multimodality among the application working modes at various levels. We have also focused on the Data-Mining Association Rule Based Learning concept to introduce new service as composition of two or more services and thereby reducing the number of client requests to handle.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Patterns; D.2.10 [**Software Design**]: Methodologies

Keywords

Autonomic Computing System, Design Patterns, Aspect-Oriented Programming Design Pattern, Aspect-Oriented Programming (AOP), Remote Method Invocation (RMI), Java Aspect Components (JAC), Feature-Oriented Programming (FOP).

*Part-time Research Scholar, Department of Computer Science and Engineering, National Institute of Technology, Warangal, INDIA. email: vishnumannava@gmail.com.

CCSEIT-12, October 26-28, 2012, Coimbatore, [Tamil Nadu, India]

1. INTRODUCTION

The most widely focused elements of the autonomic computing systems are self-* properties. So for a system to be self-manageable they should be self-configuring, self-healing, self-optimizing, self-protecting and they have to exhibit self-awareness, self-situation and self-monitoring properties [3]. The vision of autonomic computing [20] is to cut down the configuration, operational and maintenance costs of a distributed system by enabling systems to provide the self-reconfiguration, self-manage properties. So in order to achieve the vision of an autonomic computing system, it requires a system to be able to dynamically adapt to its environment and most of the adaptations that are used in an autonomic system would tend to be crosscutting in nature. Different programming paradigms have been introduced for enhancing the dynamic behavior of the programs. Few among them are the Aspect oriented programming (AOP) and Feature oriented programming (FOP) with both of them having the ability to modularize the crosscutting concerns, where the former is dependent on aspects ,advice and later one on the collaboration design and refinements.

Design patterns yields better-quality software within reduced time frames. When designing software two or more patterns are to be composed to solve a bigger problem. Pattern composition has been shown as a challenge to applying design patterns in real software systems. composite patterns represent micro architectures that when glued together could create an entire software architecture. Thus pattern composition can lead to ready-made architectures from which only instantiation would be required to build robust implementations [2]. A composite design patterns shows a synergy that makes the composition more than just the sum of its parts [21]. The AspectJ which is mainly concerned about the creation of the Aspect-Oriented Programs, and the AspectJ is a language that defines new keywords. In AOP the crosscutting concerns are handled in separate modules known as aspects. And it is very popular programming to separate the crosscutting concerns from the general Object Oriented Code. So by this type of separation of the main concerns into aspectual modules, we have the power of modularizing the crosscutting concerns as the separate modules. We can attain the dynamic reconfiguration in the applications with the help of dynamic crosscutting up to some extent.

There are other Frameworks out there in the market that provide the dynamic reconfiguration in run-time, one among them is the Java Aspect Component (JAC). JAC [16] provides an API that allows you to define aspects, Pointcuts, and advice code. When it comes to the aspect weaving mechanisms, AspectJ Weaves at compile time or at load time.

Whereas the JAC weaves aspects at run-time, which leads to more adaptable programs because aspects can be dynamically added and removed at run-time.
The Aspect Configuration is an essential part when programming with JAC. So this feature is not provided by the AspectJ. The Aspect-Configuration File plays a very crucial role in the development reusable aspects in JAC. The concept of Aspect Configuration File exists for both J2EE and the JAC, but there is a difference in the way that they both deal with the configuration process. A set of parameters is associated with the persistence, Transaction, naming, access-control service. When it come to J2EE these parameters are fixed and provided by the servers and cant́ be changed. With JAC we can choose precisely the configuration parameters that we provide to provide.
Each and every aspect defined with JAC is associated with an Aspect Configuration File that actually saved with an .acc extension. This file is loaded at run-time when the aspect is instantiated and this file can be modified and reloaded while programming is running. So with this the configuration of the program is then dynamically adapted.
The aspect-configuration file provides the current values for the definition of the pointcut. These values can be changed without recompiling the aspect. Reloading the aspect configuration file into JAC at run time causes the pointcut that is associated with the aspect and the traced join points to change. Each JAC application is associated with a .jac file. This is file provides the information about the entry point of an application and the aspects that need to be initially woven.
We use the Autonomic Computing System model introduced by Ramirez in [18] for the development of an application that provides the reconfiguration among the service providing aspects in a server.
Self-Healing plays a crucial role when the server goes down or there is a dropdown in the service provided by a server due to over load of requests received from clients. So in order to overcome this drawback we have provided the self-healing capability with the help of Java Aspect Components (JAC), Java Web Services (JWS) and JUDDI Service Repository.

2. RELATED WORK

In this section we present some works that deal with different autonomic systems design. There are number of publications reporting the adaptive nature of the systems where changes occur depending upon the environments in which they are deployed. They provide the ability to monitor, to make decisions and to reconfigure at run-time.
In Rasche and Poize paper [19] they will analyze the timing behavior of the implemented dynamic reconfiguration algorithm in order to allow for predictable execution times. They describe how complex component-based real-time applications can be adapted to changing environmental conditions, continuously meeting all tasks deadlines during dynamic reconfiguration.
In Olivier Aubert, Antoine Beugnard [1] they proposed an Adaptive Strategy Design Pattern that can be used to analyze or design self-adaptive systems. It makes the significant components usually involved in a self-adaptive system explicit, and studies their interactions. They show how the components participate in the adaptation process, and characterize some of their properties.
In M.Vishnuvardhan and T.Ramesh paper [8] discuss applying the Adaptive Monitoring Compliance Design Pattern for autonomic systems. The authors of the paper uses adaptive design pattern called adaptive sensor factory have been proposed to make the monitoring infrastructure of the adaptive system more dynamic by fusing the sensor factory pattern, observer and strategy patterns. This pattern will determine the type of sensor that suits best for monitoring the client.
Harald Psaier et al. [17] they introduce a self-healing approach enabling recovery mechanisms to avoid degraded or stalled systems. There presented work extends the notion of self-healing by considering a mixture of human and service interactions observing their behavior patterns. They present the design and architecture of the VieCure framework supporting fundamental principles for autonomic self-healing strategies. They validate their self-healing approach through simulations. The paper [9] they have proposed a design pattern for Autonomic Computing System which is designed with Aspect-oriented design patterns and they have also focused on the amalgamation of the Feature-oriented and Aspect-oriented software development methodology and its usage in developing a self-reconfigurable adaptive system.
The paper [15] they have proposed a system for dynamically configuring communication services. Server will invoke and manage services based on time stamp of service. The system will reduce work load of sever all services in executed by different threads based on time services are executed, suspended and resumed. The paper [13] they have proposed an adaptive reconfiguration compliance pattern for autonomic computing systems that can propose the reconfiguration rules and can learn new rules at runtime. The authors in paper [9] have mainly concentrated on providing the adaptability to the application at runtime by using Aspect-Oriented Programming. So here we would like to focus on one drawback that the authors may face with AspectJ. AspectJ when compared to Java Aspect Components (JAC) will weaves the aspects at compile time or at load time. Whereas JAC weaves aspects at run time, which leads to a behavior of more adaptable programs, because the aspects with JAC have the capability to be dynamically added and removed at run-time. With this we want to convey that we would like to enhance the adaptable nature of the design pattern proposed in paper [9] by applying Java Aspect Components (JAC) Framework.
In Vishnuvardhan Mannava, and T. Ramesh paper [12] discuss applying Autonomic Design Patterns for software architectures. They harvested the patterns and applied it on unstructured peer to peer networks [14] and Web services environments using Feature-Oriented Programming (FOP)and Aspect-Oriented Programming (AOP) [10].In [11] is presented a SOA based composition of web service based on user demand.

At the same time we have developed a multimodal design pattern which will address the problems in both distributed environment and among the components present with in the server environment. So, we have provide a design pattern with an amalgamation of Java Aspect Components (JAC), Aspect-Oriented Programming and Feature-Oriented Programming (FOP). Our Proposed design pattern will decide which type of service providing module have to be invoked either Distributed or with in server type of configuration module have to be invoked.

3. PROPOSED MULTIMODAL PATTERN-ORIENTED SOFTWARE ARCHITECTURE

In our proposed design pattern we mainly concentrate on the reconfiguration phase which plays the major role for providing the autonomic Self-Configuration, Self-Healing strategies. For this purpose we will provide with a simple design of the proposed KLConfig Architecture that provides the fundamental requirements to attain the self-healing and

self-configuration capability with in the server components and in distributed environment. In order to understand our work easily we will provide with a design view of the Multimodal Layered Architecture of our pattern in Figure 1.

Multimodal Layered Architecture	
LEVEL 1	This level covers the Self-Configuration and Self-Healing characteristics of an Autonomic computing system within the server components (Within the server) using Java Aspect Components (JAC), Object-Oriented and Aspect-Oriented Design Patterns.
LEVEL 2	This level covers the Self-Healing characteristic of an Autonomic computing system in the Distributed Environment using Universal, Description and Integration (UDDI) and Java Web Services (JWS).

Figure 1: The Multimodal Layered Architecture

Our work is divided into five modules. Each module provides the design pattern oriented approach to solve the Self-Configuration and Self-Healing problems and we will discuss about each and every Module in depth with the help of Class diagrams in following sections. When we combine all these modules it will result to a total complete KLConfig Architecture as shown in Figure 2. Every class diagram is represented by the separate blocks to indicate every module in our proposed KLConfig Architecture. The Modules are:

- Service Access Module
- Distributed Computing Self-Healing Module
- Monitoring Module
- Decision-Making Module
- Reconfiguration Module

3.1 Multimodal Layered Architecture

The proposed Multimodal Layered Architecture is divided into two levels. Level 1 mainly concenrate upon providing the self-healing and self-configuration among the components present within the server environment. This task is accomplished using the KLConfig Architecture. So inorder to perform the reconfiguration and healing we will make use of four modules, they are Service Access Module, Monitoring Module, Decision-Making Module, Reconfiguration Module. So here all these four modules are requiered for the purpose of providing both the self-healing and self-configuration in the appliaction/system. Initially when a service is not avilable in the server then the observer in the monitoring module will take care of informing the adpatation detector pattern in monitoring phase to do some reconfiguration work inorder to load the unloaded service Java Aspect Components (JAC) into serverś memory. This adaptation detector pattern will trigger an event to the Leader/Followers pattern in the Decision-Making phase to handle the event of loading the new service Java Aspect Component (JAC) code into server. Then the Leader/Follower will accept the event and it will select a Leader among the already pregenrated pool of threads with the help of Leader Election Distributed Computing Algorithm. In each and every thread the Case-based Reasoning pattern will be running which will select an reconfiguration plan by applying some rules to the events it got as input and then it will decide which among the set of plans is suitable for the reconfiguration of the application/system. So once a perfect plan is selected then the plan will result to the genration of Aspect Configuration File (ACF) which contains a single line describing unloaded serviceś Aspect Component code name and so with the help of the Java Aspect Components (JAC) Framework we can dynamically load the Aspect weaving code of the service into the serverś main memory in runtime resulting to the reconfiguration of the application/system.

When it comes to Level 2 in our Multimodal Layered Architecture here we will use the Distributed Computing Self-Healing Module, it will mainly concentrate upon the self-healing of the application/system in terms of distributed environment. Here when the server got an error specifying that a particular thread have been failed to invoke the service to fulfill the clients request then the observer pattern will access the UDDI repository of the services that contains the WSDL files of the services that are provided by the different servers in the distributed environment. The server will access the WSDL of the required service and then access the web service that is provided by the neighboring server in the distributed environment and then return the result of this service it gets from the neighboring server to the client that requested the service. In this way a transparency is maintained to the client when performing the self-healing operation in a distributed environment with the help of JUDDI services repository and Java Web Services (JWS).

3.2 Service Access Module

First Module is for efficient access of services at the servers we have used the design patterns for providing the reliable form of services to clients. Initially the Client will request the Centralized Service Repository to check for the availability of the service it is interested in. Then the Centralized Service Repository will check with the available services in its Repository (any database) with the help of Row Data Gateway Database access pattern for efficient database access. If the requested service is provided by a server in remote location somewhere in the network, then it will check whether it is currently activated by the respective service provider. If the field "Availability" is set to TRUE value then the Centralized Service Repository will return the status of the service as available and the Remote Object Reference through which the client can access the service from the remote location through Stubs in RMI. If the service is Inactive that is set to FALSE then it will send the message as "Service currently not available" to the client (refer Figure 3).

So once the client come to know the availability of service then initiates a connection with the remote VM containing the remote object, Marshals (writes and transmits) the parameters to the remote VM, it does not wait for the result of the method invocation instead it assigns the clients request to a new thread, which will take care of waiting for the result to be returned from the server, and so the client can do some other work instead of waiting for the result. This is what we have used in our work to provide Asynchronous RMI mechanism instead of using synchronous RMI mechanism. Then the thread will Unmarshals (reads) the return value or exception returned and then finally return the value to the caller.

Then at the server side the Skeleton will perform the initially Unmarshals (reads) the parameters for the remote method (remember that these were marshaled by the stub on the client side), then Invokes the method on the actual remote object implementation, Marshals (writes and transmits) the result (return value or exception) to the caller (which is then unmarshalled by the stub).

For providing the efficient service access to the clients we are providing the Service Access Module setup as show in the below class Diagram refer Figure 3.

3.3 Monitoring Module

In this module we mainly concentrated upon the self-healing of the applications should be down after unexpected withdrawal of a service or shutdown of the application. Here in our KLConfig Architecture, we can see the use of design patterns to support the self-healing capability for the sys-

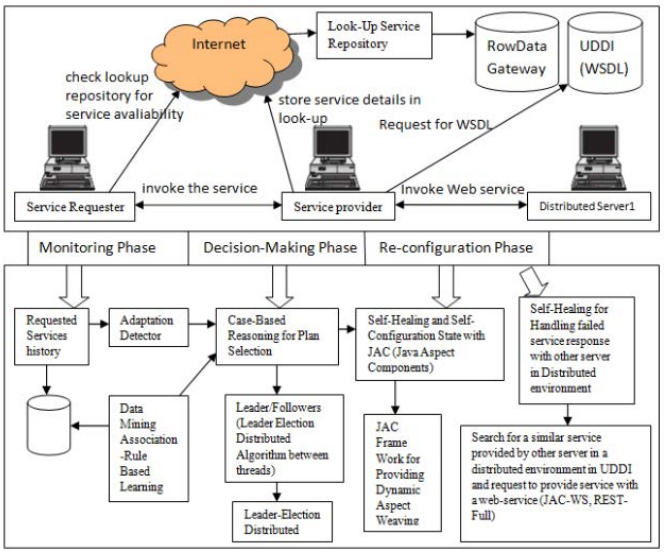

Figure 2: The proposed Design pattern and the simple design of KLConfig Architecture

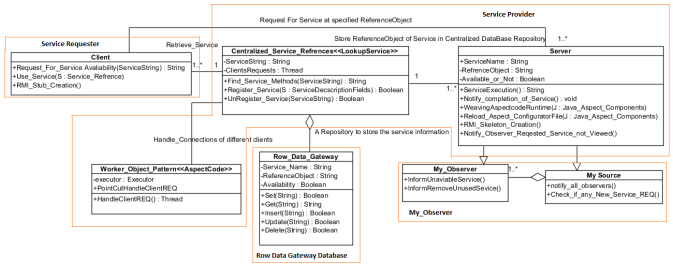

Figure 3: The Service Access Module is represented as a Class Diagram

tem. When a client request for a service, the service may or may not be available in the servers memory at present time and it have to be loaded into the main memory of the server to handle the request of the client. Most of the time all the services are not loaded into the server all at a time, but only the services that are under current use are only get loaded into the memory. With this type of practice one can expect a high end server performance with high processing speed and at the same time the rate of getting hang-up or crashing of the server due to overloaded tasks will be completely reduced. So with the help of the JAC type of frame works only the services that are currently under requirement in the server to fulfill the client requests get loaded as the Aspect components.

This Module works suitable to a server side environment. Initially when the server gets a request it will invoke the observer [5] pattern to check whether the aspect weaving code to fulfill this service is already available at the main-memory of the server. If the program is well in a swing and already ready for the server to serve the clients request no need to worry at all. But if the service is not available at the server side at present in the main memory, then it is the time to provide some self-reconfiguration capability in the system to load the unavailable service into the memory of server without stopping the server from doing already running transactions.

To do this task initially the observer will store all the information regarding the requested services by the same client into a Requested Services History either the service available in server's main memory or not. After that the Observer pattern will invoke the Adaptation Detector design pattern to generate an appropriate event that best matches appropriate handler in the Leader/Followers pattern for the service requirements given as input to the adaptation detector pattern by the observer pattern. Then the event gets generated from the adaptation detector pattern and which intern is given as input to the Case-Based reasoning design pattern.

In order to provide the Self-Configuration and Self-Healing between Server components, we are providing the Monitoring Module setup as show in the below Class Diagram refer Figure 4.

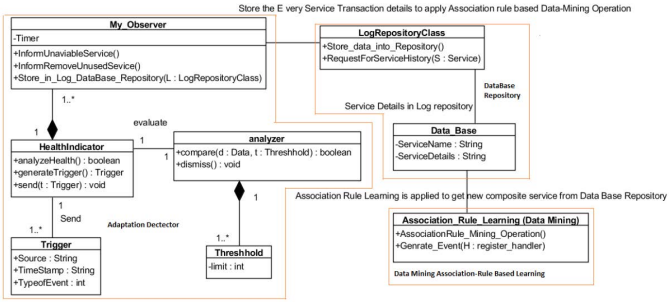

Figure 4: The Monitoring Module is represented as a Class Diagram

3.4 Decision-Making Module

We are using the Leader/Followers design pattern in which a pool of threads will be already created, where each thread have to handle an event generated by the adaptation detector pattern. So in this pattern only one thread will act as a leader at any time and all the others will act as followers and they will be sleeping until there chance comes to become leader. Once the leader thread gets an event to handle then it will inform the other followers to elect their new leader among themselves again with the help of Leader-Election Distributed Algorithm (Refer Figure 5). So here in each and every thread in this pool the Case-Based Reasoning design pattern will be running, Which intern gives the solution with a New Reconfiguration plan for a particular event that has been generated at that time. So here each and every thread in the leader/followers pattern will handle the same kind of event or different kind of event. For example here in our case we want the unloaded service to be loaded into the memory, so to do that we need the perfect reconfiguration plan from the case-based reasoning deign pattern. Here the plan is given as a new Aspect Configuration File that actually saved with an .acc extension. This file is loaded at run-time when the aspect is instantiated and this file can be modified and reloaded with the requested service while programming is running. So with this the configuration of the program is then dynamically adapted. In order to provide the Self-Configuration and Self-Healing between Server components, we are providing the Decision-Making Module setup as show in the below Class Diagram refer Figure 5.

3.5 Reconfiguration Module (Self-Healing Capability)

We can also provide the self-healing capability with the help of JAC (Java Aspect Components). When the server is filled up with some services and there is no space for the new service to get into then it will lead to failure in providing the requested service, even that requested service is available in repository, but no space in server to load it. So in order to overcome like this situation, the observer pattern will set timers for the already loaded services in the servers main memory, after the timer expires the observer will check for the count value that the service has be accessed and the

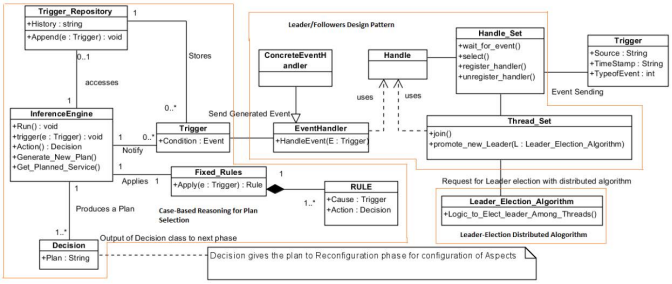

Figure 5: The Decision-Making Module is represented as a Class Diagram

last time the service was accessed and also loaded. It will send this data to the adaptation detector pattern to check if there is a need to remove this service from the memory. If "yes" then this pattern will generate a trigger event and this will be given as input to the case-based reasoning pattern to select a perfect plan (Aspect Configuration File) to deactivate the service from the main memory of the server and give chance to the new service to get in. So like this we can achieve the self-healing capability in the system.
The Class Diagram Representation of Reconfiguration Phase of the Architecture can be seen in Figure 6.

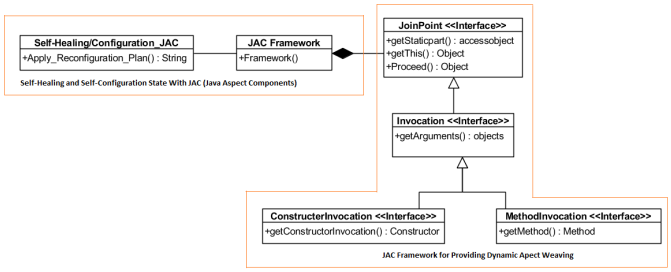

Figure 6: The Reconfiguration Module is represented as a Class Diagram

3.6 Distributed Computing Self-Healing Module

Here we want to discuss a situation that when we are performing the Reconfiguration, there may be a possibility that all the services that are currently loaded in the main memory of the server are under processing the transactions and there is no chance of removing any of the services from the server to load a new Service Java Aspect Component or there may be a possibility that the task assigned to the server thread has been failed or Maximum count of the threads that can be assigned for processing the client requests have been reached its threshold value. Then under that circumstances there must be a quick remedy for such failure happening transactions, so in order to provide self-healing for those uncompleted transactions we have to use the Distributed Computing methodology. Here what we do is, initially when the server is run out of its performance or when all the services in the servers main memory are used for longer transactions then we have to make sure that the server is not congested with the clients requests. To overcome this we will be using the services that are provided by the other servers in the distributed environment and this could be done with the help of Java Web Services like JAX-WS or REST-FULL web services (refer Figure 7). When a server want to access a service then it will first check the JUDDI repository to get the perfect WSDL file that is well matched to the output that service providing server will result to the given input parameters of the service requesting server. Once a perfect match is found then the server will get the WSDL details about that service providing neighboring server and then it will generate a WSDL request accordingly and send it to the service providing server to process the clients request on behalf of it. Once the processing of the request is completed then the result is send back to the respective requested server, which intern return the result back to the client that requested it to provide the service. So with this method of self-healing from failures we can provide maximum transparency to the client from knowing the problem at the server and providing him with sure service guarantee.

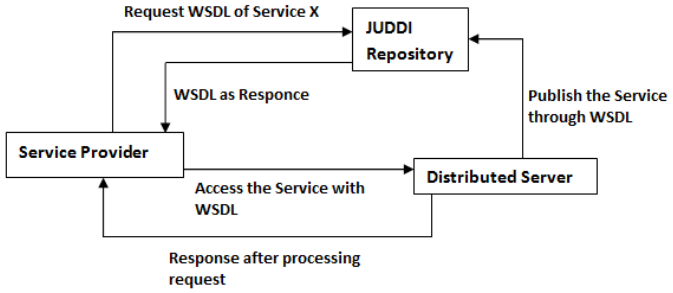

Figure 7: The Distributed Computing Self-Healing with JWS

4. THE PROPOSED ALGORITHMS FOR SELF-CONFIGURATION AND SELF-HEALING

4.1 Self-Reconfiguration Algorithm

Require: Monitoring of all currently available services.
Require: Listening to client Requests.
(variables)
Boolean flag<-false
Integer threads<-10
Boolean participate<-false
String plan<-NULL
Integer Timer
Integer Threshold<-10000ms
String ServiceName<-clientRequest
Struct leaderThread
Integer leaderID
Event event
Probe integerIDofthread
Selected electedleaderID
(1) when a client Requests for service in server
(2)ServiceName<-clientRequest
(3)send to flag=OBSERVER(ServiceName)
(4)if flag=TRUE
(5)then IGNORE //Means service is already
loaded in memory no need of reconfiguration.
(6)else send to flag=RECONFIGURATION(ServiceName)

(7)if flag=TRUE
(8)execute print ”Reconfiguration Successful,
(9)service loaded successfully”
(10)check for any Association Rule
(11)Based Generated Event
(12)Event=AssociationRuleLearning();
(13)if Event!=NULL
(14)execute generate(Event)
(15)CheckIfServiceLoaded Action OBSEVER(ServiceName)
(16)for i=1 to sizeof(Listalreadyloaded)
(17)if ServiceName = Listalreadyloaded[i];
(18)setTimer(ServiceName)
(19)return TRUE
(20)LogTransactions(ServiceName)
(21)else
(22)return FALSE
(23)LogTransactions(ServiceName)
(24)endif
(25)Self-Configuration Action
RECONFIGURATION(ServiceName)
(26)LogTransactions(ServiceName)
(27)flag<-compare(ServiceName,Threshold)
(28)if flag=TRUE
(29)flag=generate(Trigger)
(30)if flag=TRUE
(31)return TRUE
(32)Trigger Action generate(Event)
(33)poolofThreads[Threads]
(34)LeaderThread.leaderID=LeaderElection(poolofThreads)
(35)leaderThread.event=Event
(36)plan=RunleaderThread(leaderThread)
(37)flag=DispatchplanTOJAC(plan,ServiceName)
(38)if flag=TRUE
(39)execute print ”Service Loaded Successfully”
(40)else
(41)execute print ”ERROR in loading” (42)return flag
(43)LeaderElect Action LeaderElection(poolofThreads)
(44)participant<-false // becomes true
(45)when pool of threads is participating
(46)when a thread wakes up to participate in leader election
(47)sendPROBE(i) to right neighbor
(48)participate<-true
(49)when a PROBE(k) message arrives
(50)from the left neighbor poolofThreads
(51)if participant=false then execute step [43] first
(52)if i>k then
(53)discard the PROBE
(54)else if i<k then
(55)forward PROBE(k) to right neighbor
(56)else if i=k then
(57)declare I is leader
(58)circulate Selected(x) message arrives from left neighbor
(59)if x!= I then
(60)vote x as the leader and forward message to right neighbor
(61)else do not forward the selected message
(62)return selected (i)
(63)RunLeader Action RunLeaderThread(leaderThread)
(64) selectaplanfor(leaderThread.Event)
(65) return Plan
(66) JACReconfig Action DispatchPlanToJAC(plan)
(67) if plan!=NULL then
(68) ApplyJACFramework(plan,ServiceName)
(69) endif
(70) setTimer(ServiceName)
(71) Return TRUE

4.2 Self-Healing Algorithm

Require: Monitoring of all currently
available services.
Require: Listening to client Requests.
Boolean flag<-false
Boolean participate<-false
String ServiceName<-clientRequest
String ServiceName1<-NULL
Boolean modify<-false
String WSDL<-NULL
(1)when a client Requests for service in server
(2)ServiceNameïČ§clientRequest
(3)send to flag=OBSERVER(ServiceName)
(4) if flag=TRUE
(5) then IGNORE
//Means service is already loaded
in memory no need of reconfiguration.
(6) else
(7) if ServerMemory=FULL
(8) then execute self-heal()
(9) flag=RECONFIGURATION(ServiceName)
(10)if flag=TRUE
(11) execute print ”service loaded successfully”
(12)heal Action self-heal(ServiceName)
(13)ServiceName1= getserviceNametoRemove()
(14)if ServiceName1=NULL
(15) then switchtodistributedMode()
(16)else
(17)flag=ApplyJACFrameworktoheal(ServiceName)
(18)endif
(19)if flag=TRUE return TRUE
(20)healing Action ApplyJACFrameworktoheal(ServiceName)
(21)Boolean modify=
modifyAsoectConfigurationFile(ServiceName)
(22)if modify=TRUE
(23)then execute print ”the selfheal is perfect”
(24)return TRUE
(25)DistributedComputing Action SwitchtodustributedMode()
(26)WSDL= finda serviceinUDDI(ServiceName)
(27)if WSDL!=NULL then
(28) invoke the service at WSDL
(29)else
(30)search for new invoke again find
a service in UDDI(ServiceName)
(31)endif

5. DESIGN PATTERN TEMPLATE

To facilitate the organization, understanding, and application of the proposed design patterns, this paper uses a template similar in style to that used in [18].

5.1 Pattern Name

Multimodal Pattern Oriented Software Architecture for Self-Healing and Self-Configuration.

5.2 Classification

Reconfiguration Design Pattern.

5.3 Intent

Systematically applies the Design Patterns to a distributed Computing System to provide reliable access to the services at remote servers and for providing the dynamic reconfiguration using KLConfig Architecture.

5.4 Context

Our design pattern may be used when:

- The service that you want to access is on a remote system and you need to invoke it with the help of design patterns and efficient access to database using database access related patterns.

- When required to perform the reconfiguration at the server side between the Java Aspect components present with in a server or distributed in a network.
- To perform the dynamic reconfiguration without much delay.
- To provide the self-healing capability to our application to load unavailable services.

5.5 Proposed Pattern Structure

A UML class diagrams for the proposed design Pattern can be found as different Modules in Figures 3 and 4.

5.6 Participants

- **Client:** It is Responsible for the purpose of accepting the client request and then generating a request for the checking of availability of a service in the Centralized Service Reference. If the return result is true then the service is available at a particular server and it will receive the Reference object to access that service through RMI.
- **Centralized Service References (LookupService):** It is responsible for the purpose of finding whether the required service is available at any server by checking in its Data base repository using Row Data Gateway. Here each and every server will store the details of the services it provides in the Data Base Repository through Centralized Service Reference. Each and every server will provide the entry values in the data repository for Name of the service, Reference Object to access it, and Availability status of it currently.
- **Row Data Gateway:** is responsible for the purpose of providing the efficient access to the database using Centralized Service References.
- **Data Base Repository:** it is the database where all the information about a service that is available at the server is provided in form of attributes in a database.
- **Worker Object Pattern (AspectCode):** it is the design pattern responsible for the purpose of handling the requests from the multiple clients in separate worker object threads. So that the Centralized Service Repository will not be going into a waiting state to handle another client request until the first client is serving is completed.
- **Server:** It is responsible for accepting the client requests and then invoking the observer to check that the service is available with the server to weave the aspect component code to fulfill the service requested.
- **Observer:** This is responsible for the purpose of checking whether the service is already loaded into the server′s main memory and if not then it will invoke the Adaptation detector pattern to load the service into the main memory with the help of JAC framework.
- **Health Indicator:** This class is responsible for the propose of checking whether there is a need to generate an event and to check that whatever the timer details of a particular service are send by the observer will be checked by the threshold value and if there is a need to remove that service form the servers main memory then the event will be triggered by this class which will be given as input to the case-based reasoning class.
- **Handle Set:** This class is responsible for waiting for an event to occur and selects a handler from the available thread handlers in the pool of threads.
- **Thread Set:** This class is a collection of the threads where each thread is responsible for handling different events that′s get generated either from a Association Rule based learning or from Adaptation Detector pattern.
- **Event Handle:** This class is responsible for providing the perfect reconfiguration plans to be implemented to attain the self-healing capability. This is achieved with the help of case-based reasoning pattern. This pattern is running in every thread in the thread pool and any event that is dispatched to any one of this threads, then that thread will provide use with a best reconfiguration plan or self-healing plan or new service (composite service) insertion plan at run time with the help of Aspect Configuration File of JAC (Java Aspect Components).
- **Trigger:** is the class that takes output of the adaptation Detector pattern as the input of the case-based reasoning pattern. Here then it will pass it on to the inference engine to check for the available plans.
- **Inference Engine:** this will select a perfect plan with the help of the fixed Rules class and will use the output of this class as the perfect decision to implement a self-healing or self-reconfiguration or new composite service injection.
- **Decision:** This class is responsible for providing the perfect Aspect configuration File to which will lead to more adaptable programs because aspects can be dynamically added and removed in a Java Aspect Component (JAC).

5.7 Consequences

- With the help of this design pattern we can efficiently access the services in the remote locations very comfortably by using database access pattern.
- With the help of our proposed KLConfig Architecture, we can achieve the Dynamic Reconfiguration among the java Aspect components in an Autonomic computing system.

5.8 Related Design Patterns

- **Divide and Conquer Design Pattern [18]:** This pattern can be used to determine the specific sequence of steps required to safely perform a reconfiguration. Whenever a step requires that a component be removed, it can be carried out by the Component Removal [18] Pattern.
- **Architecture-Based Design Pattern [18]:** This design pattern can be used to represent a system and its reconfiguration plans as architectural models. Models that satisfy the adaptation requirements indicate how the system should be reconfigured.

6. CONCLUSION AND FUTURE WORK

In this paper we have proposed a design pattern which provides the efficient service access for the clients using the look-up design pattern and row data gateway design patterns by Asynchronous RMI model of distributed computing. In this work we have mainly concentrated on providing the Self-Reconfiguration, Self-Healing properties of the autonomic computing system with the help of both object oriented and aspect oriented design patterns. We have proposed simple framework design called KLConfig Architecture which mainly uses the JAC to provide the dynamic aspect code insertion and deletion at run-time. There is a huge scope for the future work; we aim to implement the same

design pattern for the service composition using the Service Oriented Architecture (SOA). And also we are studying how to apply the same logic of reconfiguration of the components in a Peer-to-Peer based distributed system.

7. RESULTS AND DISCUSSION

Numbers of GoF Design patterns are usually applied in a composed form with each other in many applications, and it was abstracted from pattern realizations in several real system implementations [4]. However, none of the researchers have given the metrics upon pattern compositions [6]. In the literature Nelio Cacho at al. [2] presented the measurement results for the 62 compositions, but the evaluation of compositions involving not more than two patterns. Most of these compositions are documented through the GoF pattern catalogue. It is seen that the application of both design metrics and design patterns is geared to a common purpose, namely the elimination of bad design practices. Formally it is proved that non pattern form of software design is less efficient compared to pattern form in terms of design metrics [6]. However, we collected metrics using Analyst4j[1] an Eclipse IDE plug-in and compared the non pattern form of the proposed composite patterns with the corresponding pattern form. we can infer that pattern form of the proposed composite patterns code is good it agrees with the results in [7] [2].

8. ACKNOWLEDGMENTS

This work has been supported in part by Faculty Research Allowance Program from K L University (Koneru Lakshmaiah Education Foundation), India.

9. REFERENCES

[1] O. Aubert and A. Beugnard. Adaptive strategy design pattern, 2001.

[2] N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista, and C. Lucena. Composing design patterns: a scalability study of aspect-oriented programming. In *Proceedings of the 5th international conference on Aspect-oriented software development*, pages 109–121. ACM, 2006.

[3] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the vision of autonomic computing. *IEEE Computer*, 43(1):35–41, 2010.

[4] J. Dong, Y. Zhao, and Y. Sun. A matrix-based approach to recovering design patterns. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(6):1271–1282, 2009.

[5] R. J. E. Gamma, R. Helm and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, United States, 1995.

[6] B. Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[7] R. Lincke, J. Lundberg, and W. Löwe. Comparing software metrics tools. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pages 131–142. ACM, 2008.

[8] V. Mannava and T. Ramesh. A novel adaptive monitoring compliance design pattern for autonomic computing systems. In *ACC (1)*, volume 190 of *Communications in Computer and Information Science*, pages 250–259. Springer, 2011.

[9] V. Mannava and T. Ramesh. An aspectual feature module based adaptive design pattern for autonomic computing systems. In *Intelligent Information and Database Systems*, volume 7198 of *Lecture Notes in Computer Science*, pages 130–140. Springer Berlin / Heidelberg, 2012.

[10] V. Mannava and T. Ramesh. An aspectual feature module based adaptive design pattern for autonomic computing systems. In *Proceedings of the 4th Asian conference on Intelligent Information and Database Systems - Volume Part III*, ACIIDS'12, pages 130–140, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] V. Mannava and T. Ramesh. Composite design pattern for feature-oriented service injection and composition of web services for distributed computing systems with service oriented architecture. *International Journal of Web & Semantic Technology (IJWesT)*, 3(3):73–84, 2012.

[12] V. Mannava and T. Ramesh. Multimodal pattern-oriented software architecture for self-optimization and self-configuration in autonomic computing system using multi objective evolutionary algorithms. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pages 1236–1243, New York, NY, USA, 2012. ACM.

[13] V. Mannava and T. Ramesh. A novel adaptive re-configuration compliance design pattern for autonomic computing systems. *Procedia Engineering*, 30(0):1129 – 1137, 2012. International Conference on Communication Technology and System Design 2011.

[14] V. Mannava and T. Ramesh. A novel approach for developing jxta peer-to-peer computing systems using aspect-oriented programming methodologies. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pages 421–427, New York, NY, USA, 2012. ACM.

[15] V. Mannava and T. Ramesh. A service administrationădesign pattern for dynamically configuring communication services in autonomic computing systems. In *Intelligent Information and Database Systems*, volume 7196 of *Lecture Notes in Computer Science*, pages 53–63. Springer Berlin / Heidelberg, 2012.

[16] R. Pawlak, J.-P. Retaillé, and L. Seinturier. *Foundations of AOP for J2EE Development (Foundation)*. Apress, Berkely, CA, USA, 2005.

[17] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Behavior monitoring in self-healing service-oriented systems. In *COMPSAC*, pages 357–366, 2010.

[18] A. J. Ramirez and B. H. Cheng. Applying adaptation design patterns. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 69–70, New York, NY, USA, 2009. ACM.

[19] A. Rasche and A. Polze. Dynamic reconfiguration of component-based real-time software. In *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, WORDS '05, pages 347–354, Washington, DC, USA, 2005. IEEE Computer Society.

[20] P. Soule. *Autonomics Development: A Domain-Specific Aspect Language Approach*. Autonomic Systems. Springer Verlag, 2010.

[21] T. Taibi. Formalizing design patterns composition. *the IEE-Proceeding Software*, 153(3):127–136, 2006.

[1] http://www.codeswat.com