Chapter 2

Evolution and Paradigm Shift in Distributed System Architecture

Rahul Singh Chowhan

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.80644

Abstract

New age programming languages highlight the mobility of objects and on-the-fly communication mechanism even being available on nodes with intermittent connections. We are breathing in the era where the working framework enables the procedures to oversee imparted information and arrangement to a domain where distinctive procedures are executing on discrete frameworks that essentially makes use of message-based correspondence or mobile communication architecture. The highlight that has been conceived for the years has spawned the remote administration and remote access in distributed computing framework and was outlined as an approach to digest the strategy call component to use between frameworks associated through a system. These frameworks contains the stub and skeleton on client and server side respectively which behaves as remote proxies and deals with marshaling and unmarshalling of the incoming and outgoing data. This has incurred the need of more distributed and platform independent communication mechanisms, that can not only make intercalling of functions but also support features like platform independency from various object oriented based programming languages. The distinctions in the programming model prompt higher state of abilities and more implicit customer side mechanisms for simple and hands-on interaction with the code that actualizes and implements the distributed frameworks.

Keywords: distributed computing, active objects, remote calling, synchronous message passing, network persistent storage

1. Introduction

The distributed system architecture field is more abstract, being at the level above the algorithms and data structures fields [1]. These architectures include the global control

structures, protocols for communication, synchronization, physical distribution, scaling and performance, remote access and selection among design alternatives. At this level the common architectural styles like pipes and filters, object orientation, event-based models, and table-driven interpreter etc., and their computing techniques and distributed network architectures are seen. The level of performance within the distributed computing is directly proportional degree of multiplicity of resources involved and participating in it. This is one of the important factors that affects and regulates the usefulness within the distributed computing defining the capability of computing system to support multiplicity and migration. Most of distributed approaches involve dispersion of data over various network machines for reliability and availability of data which requires deployment of communication protocols that can server inter-platform connectivity. The interconnections in a disseminated framework allows machine to interact autonomously allowing them to share memory or processors. They can communicate with each other utilizing messages, snippets of data exchanged, intimating a function on one machine and finishing on other and so on. Simply by messages many information can be conveyed to execute with specific contentions, also they can send and get bundles of information and can do many more such things [2].

Dispersed frameworks over the network has always been communicating and imparting the information by means of message passing. This type of correspondence was exceptionally straightforward where one side (client) bundles a few information, known as a message and sends it to the opposite side where it is decoded or stored additionally. The configuration of the message and the manner by which it will be prepared by the recipient is carried out by application subordinate. In a few applications the recipient may react by sending an answer message often called as acknowledgement or response while in other cases, this won't not occur. This approach likewise makes it difficult to reuse segments of one circulated framework in other conveyed frameworks as the message is encoded in language could not be decodable or readable by other language framework with different sets of library and calling mechanisms [3]. Despite the fact that message passing can be powerful, it would be decent if there were more uniform, reusable, and easy to understand methods for getting things done remotely by calling remote function on local machine or by sending function for execution on server. Such unusualness requires an extensive variety of new procedures past those utilized as a part of conventional computing. This also involves participation of an appropriated framework by letting the compiler or run-time libraries handle various issues of scheduling and allocation.

One of the alternatives in the design of the distributed system architecture is how to access remote resources or make calls to remote objects and also how to send the program over network. Currently, the client–server paradigm is the most common style, where the code does not move at all. Under the code-mobility and remote programming domain there are certain paradigm that helps in understanding the shift happened in distributed system architecture. Like, remote method invocation allows invocation of remote objects to java enabled platforms. Code-on-demand paradigm calls upon the code from a distant site which is then downloaded and executed on the local machine. And, remote evaluation paradigm which sends the code to another site where it is executed from which result is returned back to caller [4]. These distributed mechanism helps in building up of mobile and portable code design to help effective information and program relocation in various processing states on heterogeneous execution platforms.

2. Architectural taxonomies

2.1. Centralized paradigm

Allocation of numerous resources to a small number of computers called Server-hosts, yet keeping client-hosts simpler by offloading the computation to central terminal is termed as centralized Paradigm. This type of architectural taxonomy relies heavily on network resources like servers and infrastructure for computation and storage. In this typography, client-hosts are diskless nodes that are dependent on central network terminal to load its operating system. Simply, it acts as an input/output interface to the server because they neither have their own operating system nor personalized resources. The much broader infrastructure used for such paradigm is Thin Client, which is a lightweight computer that is purposely built for remote into a server, where many client-hosts share their computations with a server or server farm. It depends heavily on another computer (server) to fulfill its computational roles. The specific roles assumed by the server-host may vary, from hosting a shared set of virtualized applications, a shared desktop stack or virtual desktop, to data processing and file storage on the behalf of client-hosts [5].

The server-side infrastructure makes use of cloud computing software such as application virtualization, hosted shared desktop (HSD) or desktop virtualization (VDI). This combination forms what is known today as a cloud based system where desktop resources are centralized into one or more data centers. Basically, this type of architecture is described by lack of delegation as they have single management station to initiate requests for low-level data.

2.2. Hierarchical paradigm

Distributed processing encompasses a wide range of task autonomy and semantic richness in hierarchical architectures. This paradigm describes implementation labels that employ vertical delegation for management functionality. Hierarchical approach includes distributed objects and limited forms of Management-by-Delegation (MbD) with code mobility technologies such as Remote EValuation (REV) and Code-on-Demand (CoD). Distributed objects describe a form of gateway operation allowing the communication with encapsulated data and actions remotely. Likewise, REV provides code for execution of intended management function while CoD retrieves and caches code to execute the intended management function [6]. The hierarchical paradigm supports the delegation as following:

a. Delegation-by-domain: Domain delegation is referred as a simplified distributed paradigm. In this, a central authority assigns complete management control of a specified domain to the domain itself. The distributed domain functions independently of the central authority. Management information is not shared, and resources and administrative control resides with the specified domain. Central authority behaves as task coordinator to delegate task to different domains.

b. Delegation by micro-task with low-level semantics: Delegation by micro task in distributed hierarchical paradigm allows the central authority to employ one or more management stations to perform specified tasks. Low-level semantics signifies the little abstraction

from the details of the management task. Likewise, this method of delegation statically retrieved low-level data from simple agents before handing the response data to the central authority for processing into information.

c. Delegation by micro-task with high-level semantics: High-level semantics refers to meaningful abstractions from low-level data. For example, this method of delegation statically retrieves object data from a distributed environment before handling the response object to the central authority for processing. This framework encapsulated the protocol that supports communication between objects. Example of this distributed object paradigm includes common object request broker architecture (CORBA) and web based enterprise management (WBEM).

d. Delegation by macro-task with low-level semantics: Delegation by macro task allowed a central authority to empower one or more management stations to control specified managed elements rather than specified element properties. The management station performs necessary functions such as statically retrieving low-level data from simple agents to be processed into information by managing application. It is also responsible for taking corrective action if central authority is lost while communications.

e. Delegation by macro-task with high-level semantics: This form of delegation involves one or more authorized management stations controlling specified managed elements. Management functions include statically retrieving object data from a distributed environment which is subsequently processed by the managing application. It allows effective control decomposition and functional approximation to promote framework scalability, run time overhead reductions and workload dynamics. Example of this approach is a Goal Driven Network Management System [7].

2.3. Cooperative paradigm

Semantically rich delegation referred to a cooperative paradigm in distributed systems that empower the remote agent to control specified elements with limited instructions for preset operations. The intelligent agent relies on high-level goals and changing contextual data to make appropriate independent determination for successful management in a complex environment. Along with high autonomy and low task specification, cooperative paradigm uses horizontal delegation to cooperate with other agents unlike vertical delegation in hierarchical approaches. This is also more effective for real-time data collection within large complex and evolving networks. However, these approaches require some sort of system fidelity and measures of consistency across all nodes ensuring cooperation towards a common goal [8].

3. Client-server architecture

Distributed application structure defines client–server model that does segregation of workloads between service or resource provider, called servers and service or resource requester, called clients. These two separate components, a client and a server, which communicate over

http://dx.doi.org/10.5772/intechopen.80644

a network through a TCP/IP handshake paradigm. The client requests information, while the server responds when its advertised services are accessed. This each request/response, as depicted in **Figure 1**, is a complete round trip on the network. The code that implements these services i.e. the know-how is hosted locally by the server, also server has processing capabilities. Client decides with some intelligence which of services offered by server it should use.

3.1. One-tier architecture

Single-tier architecture is the simplest, single tier on single user, and is the equivalent of running an application on a personal computer as shown in **Figure 2**. All the components like user interface, business logic, and data storage, which are necessary to run an application, are located within the system. They are the easiest to design, but the least scalable as they are not part of a network also they cannot be used for designing web applications [9].

3.2. Two-tier architecture

Two-tier architectures supply a basic network between a client and a server. For example, the basic web model is a two-tier architecture as illustrated in **Figure 3**. A web browser makes a request from a web server, which then processes the request and returns the desired response, in this case, web pages. This approach improves scalability and divides the user interface

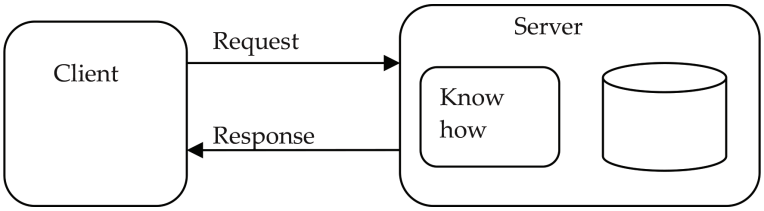

Figure 1. Client server paradigm.

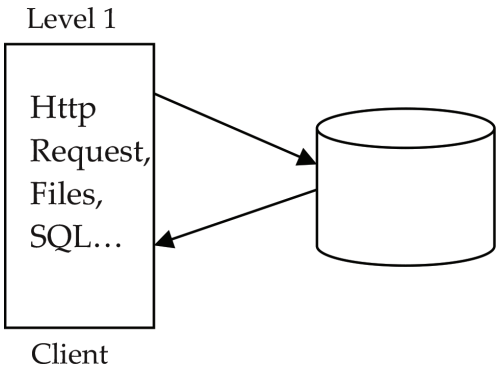

Figure 2. Single-tier architecture.

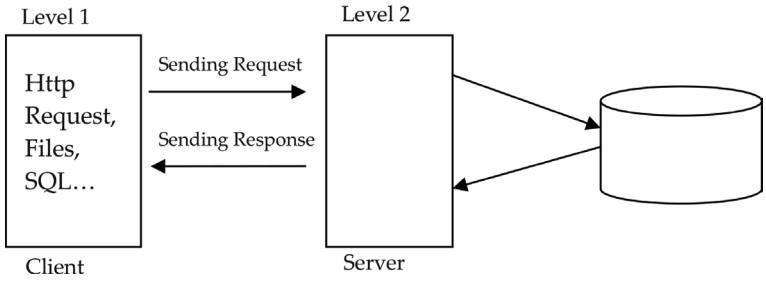

Figure 3. Two-tier architecture.

from the data layers. However, it does not divide application layers so they can be utilized separately. This makes them difficult to update and not specialized. The entire application must be updated because layers are not separated.

3.3. Three-tier architecture

Three-tier architecture is most commonly used to build web applications. In this model, the browser acts like a client, middleware or an application server contains the business logic, and database servers handle data functions. This approach separates business logic from display and data [10]. So the three layers commonly known as: presentation layer (PL/UI), business logic layer (BLL) and data access layer (DAL) as shown in **Figure 4**.

a. Presentation tier (Level 1): This provides the application's user interface (UI). Being the topmost level it displays information related to user oriented functionality responsible for managing user interaction with the system. This acts as common bridge into core business logic encapsulated in business layer.

b. Business logic tier (Level 2): This is also called application layer as it controls an application's functionality by performing detailed processing. This layer implements the core functionality of the system encapsulating the relevant business logic. It has components exposing service interfaces for callers to use.

c. Data access tier (Level 3): This includes data persistence mechanisms like database servers, file shares, etc. providing access to data hosted within system and data exposed by other networked systems. The data layer exposes generic interfaces that can be consumed by components in the business layer. It also provides an API to application layer that exposes methods of managing the stored data without out casting dependencies on the data storage mechanisms.

3.4. N-tier architecture

Terms layer and tier are often used interchangeably but one point of difference is that a layer is a logical structuring mechanism for the elements that make up the software solution. That means logical software component groups, mainly by functionality, are used for software development purpose. By contrast, a tier is a physical structuring mechanism for the system infrastructure [11].

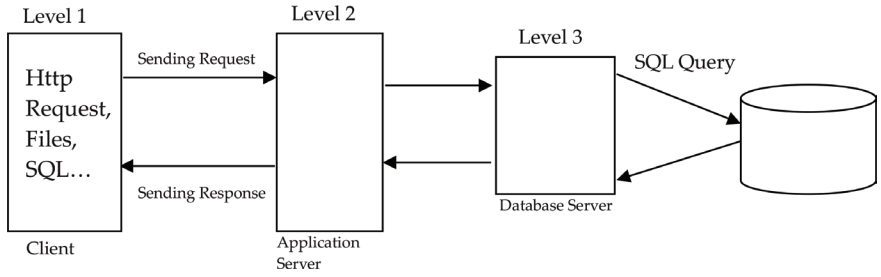

Figure 4. Three-tier architecture.

Like an individual running server is one tier and several running servers may also be counted as one tier. Layer software implementation has many advantages and is a good way to achieve N-tier architecture. Layer and tier may or may not exactly match each other. Each layer may run in an individual tier. However, multiple layers may also be able to run in one tier.

N-tier implies more than three levels or tiers involved as depicted in **Figure 5**; mostly additional tiers are associated with business logic tier. Some layers in 3-tier can be broken further into more layers. These broken layers may be able to run in more tiers. For example, application layer can be broken into business layer, persistence layer or more. Presentation layer can be broken into client layer and client presenter layer [12]. So, in order to claim a complete N-tier architecture, client presenter layer, business layer and data layer should be able to run in three separate computers (tiers).

a. Client tier: This tier is involved with users directly. There may be several different types of clients coexisting, such as WPF, Window form, HTML web page and etc.

b. Client presenter tier: This contains the presentation logic needed by clients, such as ASP. NET MVC in IIS web server.

c. Business tier: It handles and encapsulates all of business domains and logics; also called as domain layer.

d. Persistence tier: This tier handles the read/write of the business data to the data layer, also called data access layer (DAL).

e. Data tier: It is the external data source, such as a database.

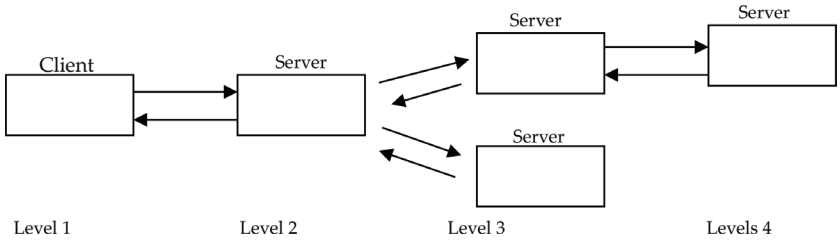

Figure 5. N-tier architecture.

4. Remote procedure mechanism

Remote procedure call works on client–server communication protocol that is used by one program to request a service from a program located in another computer in a network without understanding network details. It is based on RPC is a synchronous operation requiring the requesting program to be suspended till the results of remote procedure are returned [13].

4.1. Working and architecture of RPC

RPC is analogous to a function call extending the notion of conventional local procedure calling so that procedure need not exists in the same address space as the calling procedure. Like a function call, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure.

The client makes a procedure call that sends a request to the server and waits for response, as shown in **Figure 6**. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply back to the client. After the RPC call is completed, the client program continues its normal execution [4].

Stub: Stubs are generated at the static compilation time and then deployed to the client side which is used as a proxy for the client. Client-side proxy acts as a mediator between the client and the broker and provides additional transparency between them and the client so that a remote object appears like a local one. The proxy hides the inter-process communication (IPC) at protocol level and performs marshaling of parameter values and un-marshaling of results from the server.

Skeleton: Skeleton is generated by the service interface compilation and then deployed to the server side, which is used as a proxy for the server. Server-side proxy encapsulates low-level

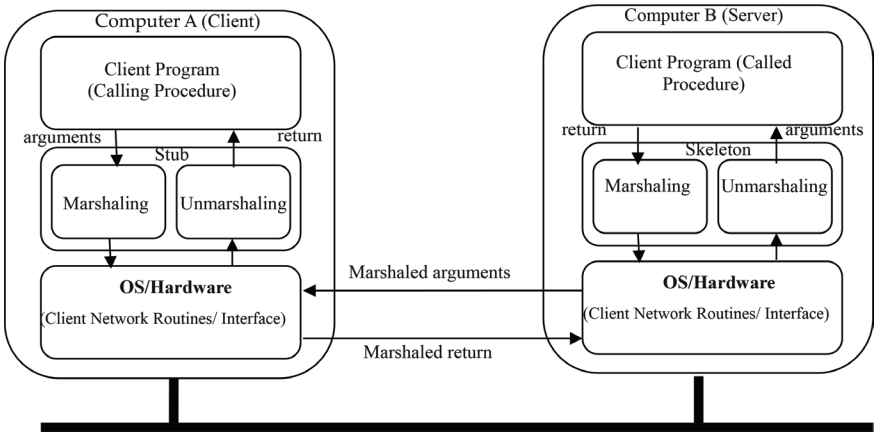

Figure 6. Remote procedure call.

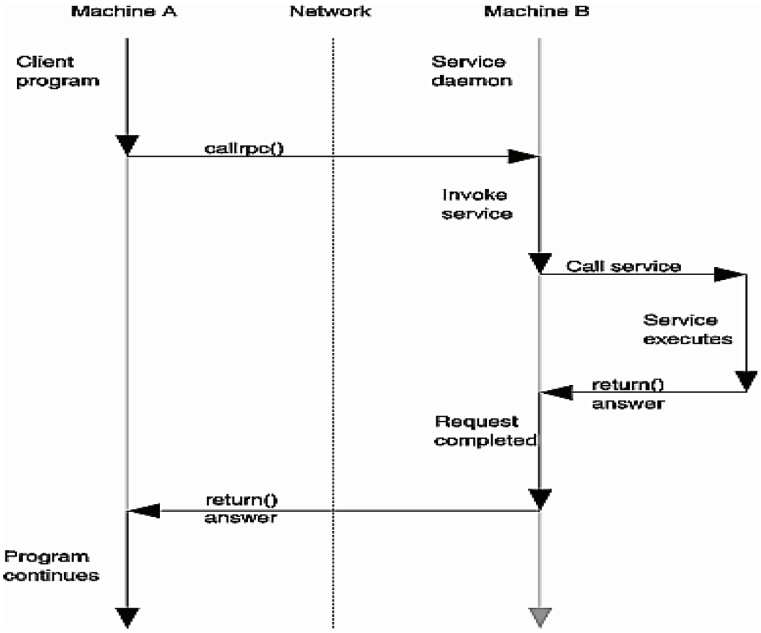

Figure 7. Event flow in RPC.

system-specific networking functions and provides high-level APIs to mediate between the server and the broker. It also receives the requests, unpacks the requests, un-marshals the method arguments, calls the suitable service, and also marshals the result before sending it back to the client [2].

Sequence of events during an RPC:

- The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshaling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called un-marshaling.

Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction. **Figure 7** shows the event flow of RPC.

5. Remote method invocation

Remote method invocation is a technology introduced by java that allows invocation of methods that are remotely located by simply calling them using desired interfaces. RMI technology allows us to distribute over business logic i.e. making the business logic available on a remote server letting it accessible to clients [14].

RMI is often called as "RPC with object orientation", i.e. the RPC but with ability to pass one or more objects along with the request. The objects can include the information that will change the service that is performed in the remote computer as delineated in **Figure 8**.

For example, when a user at a remote computer fills out an expense account, the Java program interacting with the user could communicate, using RMI, with a Java program in another computer that always had the latest policy about expense reporting. In reply, that program would send back an object and associated method information that would enable the remote computer program to screen the user's expense account data in a way that was consistent with the latest policy [15]. The user and the company both would save time by catching mistakes

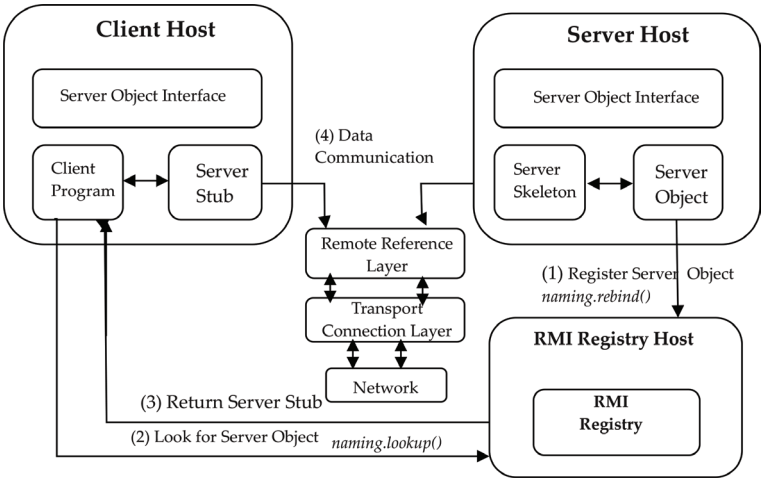

Figure 8. Remote method invocation.

RMI	RPC
Location neutral, language dependent	Language neutral mechanism
Supports object oriented design	It is procedural like C
It allows objects passing as arguments and return values	It supports only primitive data types
This allows usage of design patterns	No such capability

Table 1. RMI v/s RPC difference table.

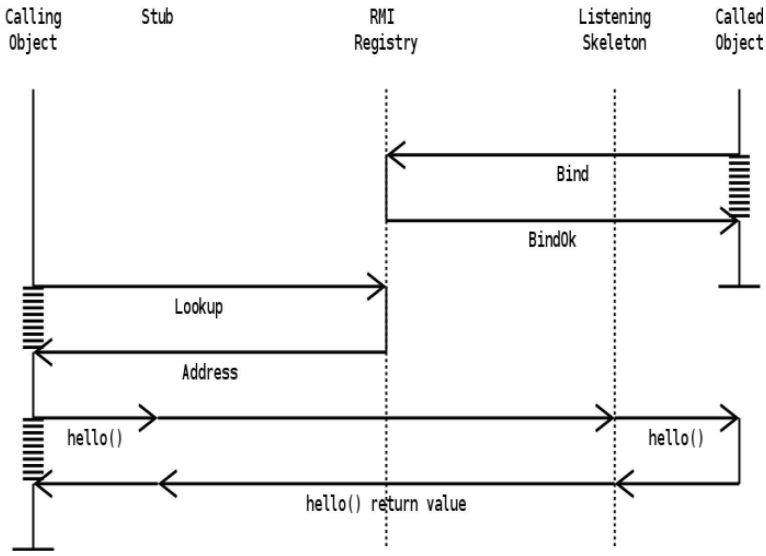

Figure 9. Event flow in RMI.

early. Whenever the company policy changed, it would require a change to a program in only one computer (**Table 1**).

RMI is implemented as three layers (as illustrated in **Figure 9**):

a. Stub/Skeleton layer: A stub program represents the remote object and also acts as gateway to a corresponding skeleton at the server end. The stub appears to the calling program to be the program being called for a service.

b. Remote reference layer: This can behave differently depending on the parameters passed by the calling program. For example, this layer can determine whether the request is to call a single remote service or multiple remote programs as in a multicast.

c. Transport connection layer: This sets up and manages the request. A single request travels down through the layers on one computer and up through the layers at the other end.

RMI Registry is a central repository keeping a track of all services being exposed from the current network. Since all the clients' requests for services through the RMI Registry the location of the application or service is unknown to the clients hence making the application location neutral [16].

6. Code-on-demand paradigm

Typically, code on demand is used for any technology that sends executable code from a server host to a client host on the request of the client's application. Code on demand is a

specific use of mobile code under the field of code mobility. In the code-on-demand style, as delineated in **Figure 10**, a client component has an access to a set of resources, but not the know-how on how to process them. It sends a request to a remote server for the code representing that know-how, receives that code, and executes it locally. So as per the code-on-demand paradigm, knowing the know-how is necessary when in need [17].

Say for example, one host (A) initially is unable to execute its task due to a lack of code (know-how). And another host (B) in the network provides the needed code. Once the code is received by A, the computation is carried out on A's machine. Host A holds the processor capability as well as the local resources. Unlike in the client–server paradigm, A does not need knowledge about the remote host, since all the necessary code will be downloaded.

Java applets are excellent practical examples of this paradigm. Applets get downloaded in Web browsers and execute locally.

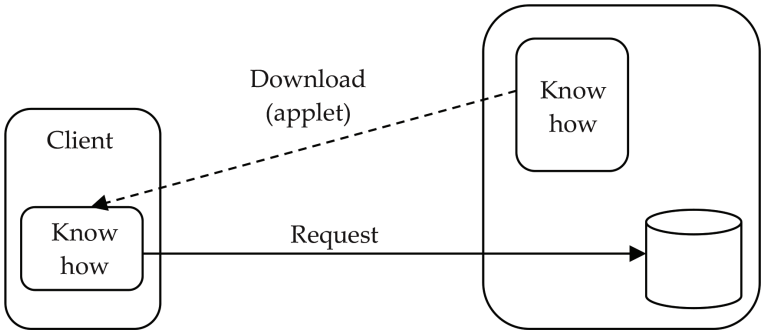

Figure 10. Code-on-demand.

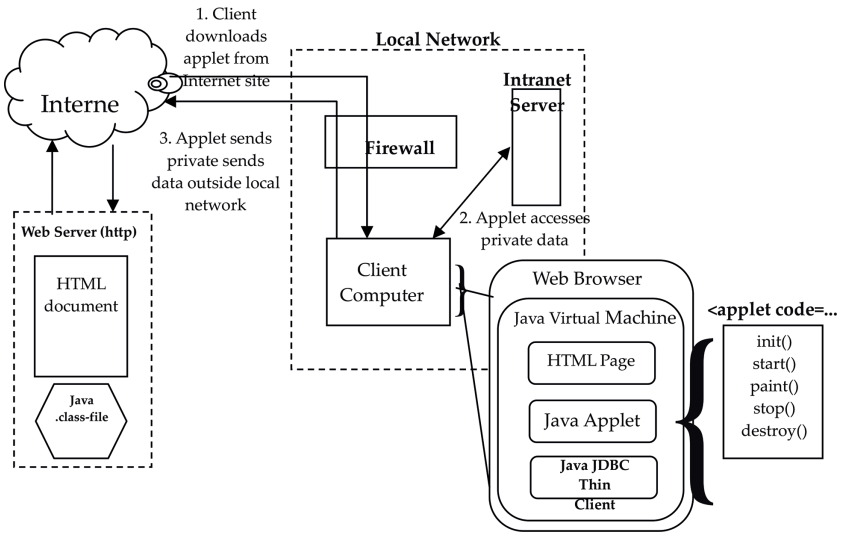

Figure 11. Architecture of applet.

6.1. Working and architecture of applets

The internet is a combination of various kinds of systems or platforms that are often required to communicate with each other. The client that makes a request may be from a completely different platform for instance the application may be hosted on the windows based server and client may be requesting from a Linux-based system.

Java introduced a new technology that would allow any client from any network platform to host and execute applications over the internet. This new technology was called as applets [18]. The word applet stands for an "application scriplets". This can be defined as a piece of java code residing on a server machine requested via a browser downloaded over the internet and executed on the client machine via the browser. In order to execute the applet on a client machine, the browser must be java enabled i.e. JRE must be enabled. An applet is typically embedded inside a web page and runs in the context of a browser. The browser's Java Plug-in software manages the lifecycle of an applet. The architecture of applet is shown in above **Figure 11**.

6.2. Life cycle of an applet

Atop these five methods, depicted in **Figure 12**, an applet is been created:

a. Init(): This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

b. Start(): This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

c. Stop(): This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

d. Destroy(): This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

e. Paint(): Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

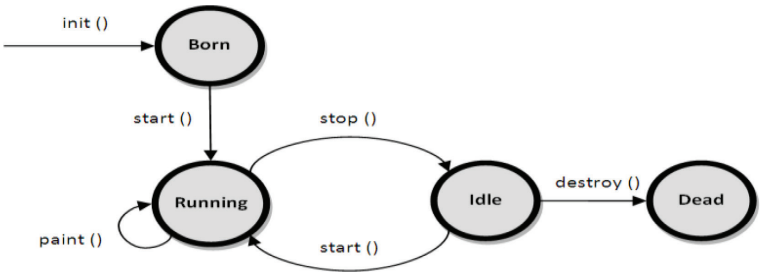

Figure 12. Life cycle of applet.

7. Remote evaluation

In computer science, remote evaluation is a term that belongs to the family of mobile code, within the field of code mobility. It is for any technology that involves the transmission of executable software code from a client hosts to a server hosts for execution to be happen at the server and the result is sent back to the client after execution for this resources of server sside are used [19]. A simple model of remote evaluation is illustrated in **Figure 13**.

An example for remote evaluation is grid computing: An executable task may be sent to a specific computer in the grid. After the execution has terminated, the result is sent back to the client. The client in turn may have to reassemble the different results of multiple concurrently calculated subtasks into one single result.

7.1. Working and architecture of servlets

7.1.1. Basic idea and architecture

Web based technologies are of two different types: Client Side Technologies and Server Side Technologies. The Client Side Technology has the code completely downloaded on the client machine and executed on the client itself, any changes that need to be incorporated or updated in the application will be on client system after re-downloading by the client. The processing of this application will take place on the client, completely.

In a Server Side Technology the complete business logic is maintained on the server and on the request of the client it will be executed on the server, delivering the response to the clients. The Java Servlets technology provides on such simple, vendor-independent mechanism to extend the functionality of a web server [20]. Servlets technology is similar to common gateway interface (CGI) scripts, Javascripts (on client side) and hypertext preprocessor (PHP). Additionally, scripting languages can be used in servlets to dynamically modify or generate hypertext markup language (HTML) pages. It also supports various HTTP methods, such as GET and POST, which is used to redirect requests and responses as shown in **Figure 14**.

7.1.2. Working and life cycle

Whenever a client sends a request to the J2EE application server for a particular servlet, the J2EE Application server passes the request to the Web container. The Web container checks

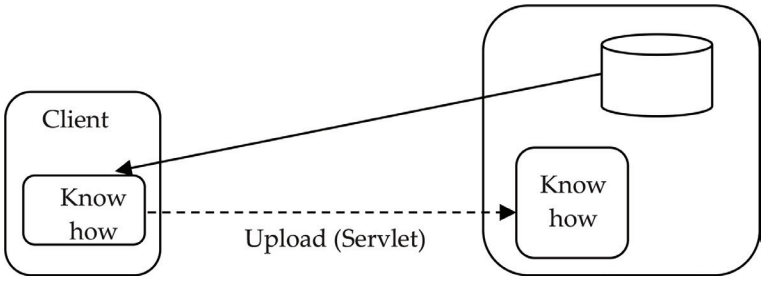

Figure 13. Remote evaluation.

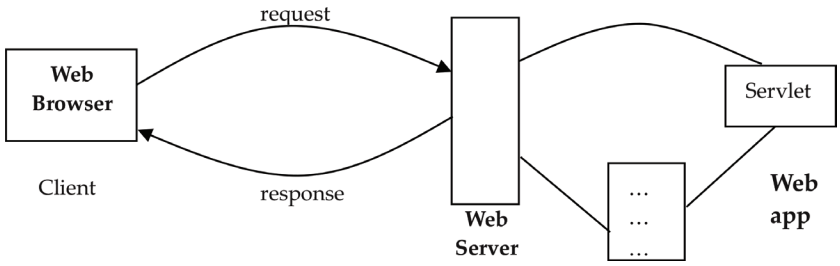

Figure 14. Architecture of servlets.

whether an instance of the requested servlet exists. If the servlet instance exists then the Web container delegates the request to the servlet, which process the client request and sends back the response (Shown in **Figure 15**).

It is the job of Web container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet. So in case the servlet instance does not exist, the Web container locates and loads the servlet class. The Web container then creates an instance of the servlet and initializes it. The servlet instance starts processing the request after initialization. The Web container passes the response generated by the servlet to the client.

Servlets don't have a main() method that's why Web container manages the life cycle of a Servlet instance. The life cycle of the servlet includes three states: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state [21]. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state. It is shown in **Figure 16**.

7.1.3. Life cycle of servlet

a. Servlet class is loaded: The class loader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

b. Servlet instance is created: The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

c. Init method is invoke: The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet.

Method Signature: public void init(ServletConfig config) throws ServletException

d. Service method is invoked: The web container calls the service method each time when request for the servlet is received [22]. If servlet is not initialized, it follows the above three steps then calls the service method. The servlet is initialized only once so if servlet is already initialized, it directly calls the service method.

Method Signature: public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException.

e. Destroy method is invoked: The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resources like memory, thread etc. **Figure 16** shows life cycle methods of servlets.

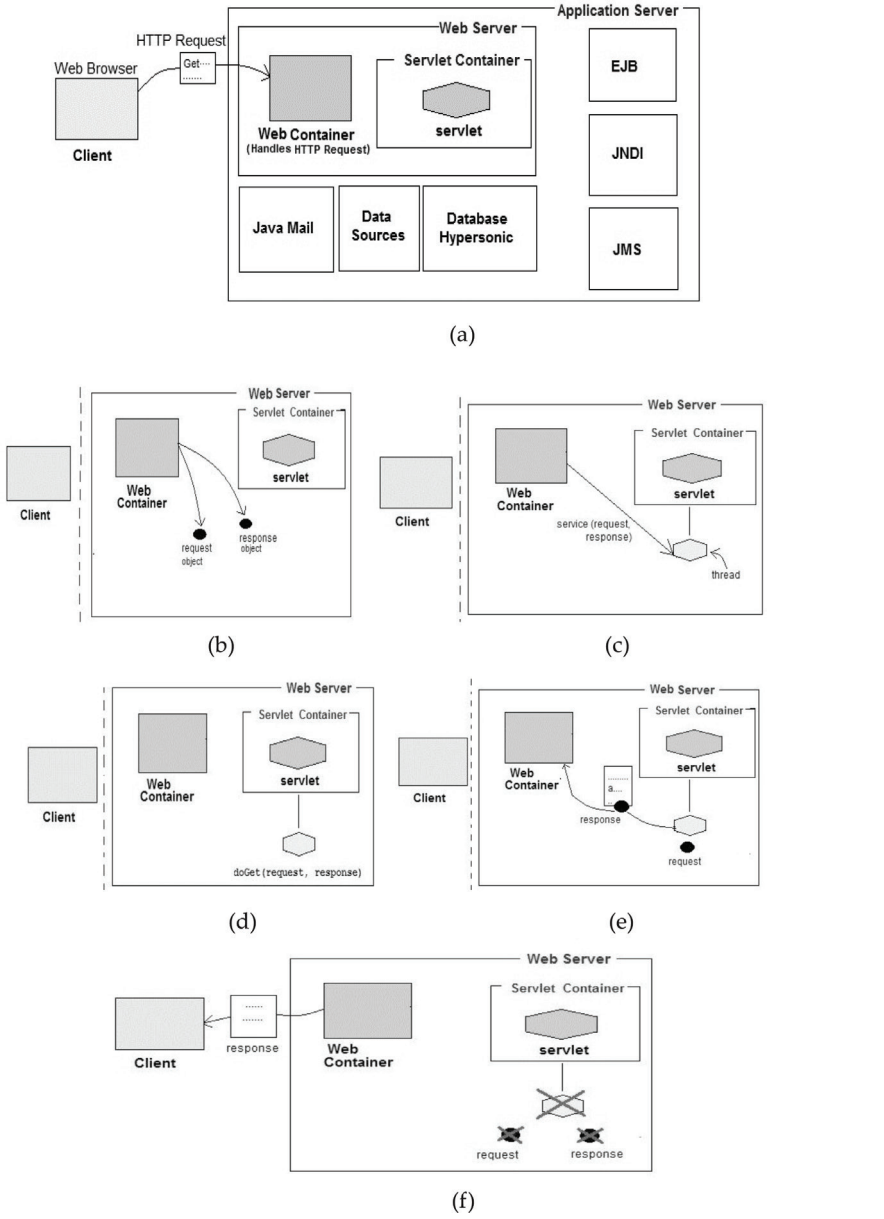

Figure 15. Various stages in request and response mechanism of servlets. (a) Clients request handling carried out by web container. (b) Object formation. (c) Calling servlet thread. (d) Thread execution. (e) Submission of response. (f) Final response toclient.

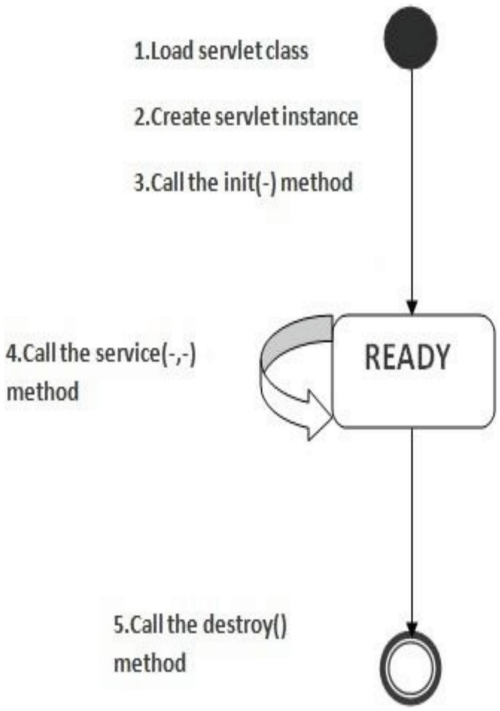

Figure 16. Life cycle of servlets.

8. Conclusion

In the past couple of years there has been a development of enthusiasm for versatile platform innovation and a few stages have been developed and innovated to allow more independencies in programming platform. In this chapter we have surveyed and researched the various computing environment provided for remote execution that has incurred the need of mobile codes, intelligent agents, autonomous objects, etc. raising issues with flexibility, efficiency and security in present system that can promises to resolve existing problems and add on more facilities like remote execution, auto-scheduling and many more. Some of them have just been utilized for look into purposes while others have been conveyed as business items. A few technologies that incorporated in evolution of mobile agents have been discussed on the basis of the usefulness of have been displayed in this exploration.

Acknowledgements

I would like to impart my sincere thanks to my guide Late. Dr. Rajesh Purohit, who has inspirited and fostered my interest in multifarious streams and disciplines of remoting and mobile-objects. Further I would like to extend my regards to all academic friends and lecturers who

supported and motivated to move on with my work. They are (alphabetical order) Ashish Sharma, Poonam Purohit, Purva Dayya and Shivam Lohiya.

The following presents the acronyms used throughout this chapter.

Abbreviations

HSD	hosted shared desktop
VDI	desktop virtualization
CORBA	common object request broker architecture
WBEM	web based enterprise management
MbD	management-by-delegation
REV	remote revaluation
CoD	code on demand
TCP/IP	transfer control protocol/internet protocol
PL/UI	presentation layer
BLL	business logic layer
DAL	data access layer
API	application programmable interface
UI	user interface
WPF	windows presentation foundation
HTML	hyper text markup language
RPC	remote procedure call
RMI	remote method invocation
Applets	application scriplets
Servlets	server scriplets
CGI	common gateway interface
HTTP	hyper text transfer protocol
PHP	hypertext pre processor

Author details

Rahul Singh Chowhan

Address all correspondence to: word2rahul@gmail.com

Agriculture University, Jodhpur, India

References

[1] Aridor Y, Lange DB. Agent design patterns: Elements of agent application design. In: Proceedings of the Second International Conference on Autonomous Agents; 1 May 1998. ACM. pp. 108-115

[2] Juziuk J. Design patterns for multi-agent systems. Linnaeus University, Faculty of Science and Engineering, School of Computer Science, Physics and Mathematics; 2012

[3] Pandey R, Sharma N, Rathore R. Aglets (A java based Mobile agent) and its security issue. International journal of emerging trends & technology in computer science (IJETTCS). 2013;**2**(4):107-114

[4] Ahila SS, Shunmuganathan KL. Overview of mobile agent security issues—Solutions. In: 2014 International Conference on Information Communication and Embedded Systems (ICICES); 27 Feb 2014. IEEE. pp. 1-6

[5] Bertsekas, Dimitri P. Dynamic programming and optimal control. Vol. 1. Belmont, Massachusetts: Athena Scientific; 1996

[6] Puterman ML. Markov decision processes: Discrete stochastic dynamic programming. Wiley Series in Probabilty and Statistics. Hoboken, New Jersey: John Wiley & Sons, Inc.; 2014

[7] Bellman RE, Dreyfus SE. Applied dynamic programming. Princeton Legacy Library. London: Oxford University Press, Princeton University Press; 2015 Dec 8:**2050**

[8] Wang D, Mu CX, Liu DR. Data-driven nonlinear near-optimal regulation based on iterative neural dynamic programming. Zidonghua Xuebao/Acta Automatica Sinica. 2017 Mar;**43**(3):366-375

[9] Ahmed K, Bigagli D, Hu Z, Wang J. Inventors; International Business Machines Corp, assignee. Resource Manager for Managing the Sharing of Resources Among Multiple Workloads in a Distributed Computing Environment. United States Patent US 9,632,827; 2017 Apr 25

[10] Kurniawan B. Java for the Web with Servlets, JSP, and EJB. New Riders Publishing; 2002

[11] Siek K, Wojciechowski PT. Atomic RMI: A distributed transactional memory framework. International Journal of Parallel Programming. 2016 Jun;**44**(3):598-619

[12] Czaja L. Remote procedure call. In: Introduction to Distributed Computer Systems. Cham: Springer; 2018. pp. 141-155

[13] Kaur M, Sharma S. A dynamic clone approach for mobile agent to survive server failure. In: 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions); 2 Sep 2015. IEEE. pp. 1-5

[14] Diaz J, Munoz-Caro C, Nino A. A survey of parallel programming models and tools in the multi and many-core era. IEEE Transactions on Parallel and Distributed Systems. 2012 Aug;**23**(8):1369-1386

[15] Hwu WM, Ryoo S, Ueng SZ, Kelm JH, Gelado I, Stone SS, Kidd RE, Baghsorkhi SS, Mahesri AA, Tsao SC, Navarro N. Implicitly parallel programming models for thousand-core microprocessors. In: Proceedings of the 44th annual Design Automation Conference; 4 Jun 2007. ACM. pp. 754-759

[16] González-Vélez H, Leyton M. A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers. Software: Practice and Experience. 2010 Nov;**40**(12):1135-1160

[17] Manogaran G, Lopez D. Health data analytics using scalable logistic regression with stochastic gradient descent. International Journal of Advanced Intelligence Paradigms. 2018;**10**(1-2):118-132

[18] Haefner JW. Parallel computers and individual-based models: An overview. In: Individual-based models and approaches in ecology. Chapman and Hall/CRC; 2018. pp. 126-164

[19] Murphy R, Sterling T, Dekate C. Advanced architectures and execution models to support green computing. Computing in Science & Engineering. 2010 Nov;**12**(6):38-47

[20] Marowka A. On parallel software engineering education using python. Education and Information Technologies. 2018 Jan;**23**(1):357-372

[21] Kemp R. Current developments in open source software. Computer Law and Security Review. 2009 Nov 1;**25**(6):569-582

[22] Coulthard P, Yantzi DJ, Simpson EV. Inventors; International Business Machines Corp, assignee. Framework to Access a Remote System from an Integrated Development Environment. United States patent US 8,296,720; 2012 Oct 23