

AN ADVANCED INFORMATION SECURITY SYSTEM TEXT-BASED GRAPHICAL PASSWORD SCHEME USING DATA MINING

**Submitted in partial fulfilment of the requirements for the award of Bachelor of
Science Degree in Computer Science**

By

MAHALAKSHMI K (Reg. No. 39290060)

SWETHA R (Reg. No. 38290108)



DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119, TAMILNADU

MARCH - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **MAHALAKSHMI K (Reg. No.39290069)** and **SWETHA R (Reg. No.3929108)** carried out the project entitled **"AN ADVANCED INFORMATION SECURITY SYSTEM TEXT-BASED GRAPHICAL PASSWORD SCHEME USING DATA MINING"** under my supervision from November 2021 to May 2022.

Internal Guide

Dr. M. SELVI, M.E., Ph.D.,

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.,

Submitted for Viva voce Examination held on

Internal Examiner

External Examiner

DECLARATION

I, **MAHALAKSHMI K (Reg. No. 39290060)** hereby declare that the Project Report entitled

“AN ADVANCED INFORMATION SECURITY SYSTEM TEXT-BASED

GRAPHICAL PASSWORD SCHEME USING DATA MINING”

done by me under the guidance of **Dr. M. SELVI, M.E, Ph.D.**, is submitted in partial

fulfilment of the requirements for the award of Bachelor of Science degree in

Computer Science.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D.**, Dean, School of Computing and **Dr. S. VIGNESHWARI, M.E., Ph.D.**, and **Dr. L. LAKSHMANAN, M.E., Ph.D.**, Head of the Department, Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. M. SELVI, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways For the Completion of the project.

ABSTRACT

In this system a spelling based Puzzle system, named and password spell.

The proposed scheme, we present a new security primitive based on hard AI problems Click spell, not only tries to improve both security and usability of Puzzle, but also aims to expand the applicability of Puzzle CAPTCHA. Click spell attempt to achieve following goals:

- ✓ Security: Puzzle Login using bubble sort.
- ✓ Usability: To increase the rate of passing the test.
- ✓ Extensibility: To provide a dictionary function for users to learn about the meaning and the spelling of words. In addition, an advertisement can be placed on the top of CAPTCHA images.
- ✓ Security: Image Puzzle solving Using Advanced Encryption Standard (Aes) Algorithm
- ✓ OTP Generation.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
1	OBJECTIVES	1
	1.1 SYNOPSIS	1
	1.2 EXISTING SYSTEM	1
	1.3 DISADVANTAGES OF EXISTING SYSTEM	2
2	PROPOSED SYSTEM	2
	2.1 ADVANTAGES OF PROPOSED SYSTEM	3
3	SYSTEM ARCHITECTURE	4
	3.1 HARDWARE	5
	3.2 SOFTWARE	5
4	LITERATURE SURVEY	5
	4.1 DATA FLOW DIAGRAM	7
5	MODULES	16
	5.1 PUZZLE LOGIN	16
	5.2 RANDOM CAPTCHA SELECTON	17
	5.3 IMAGE PUZZLE LOGIN	17
	5.4 OTP GENERATION	18
	5.5 ONLINE BANK	18

6	ALGORITHM	21
	6.1 DEFINITION	21
	6.2 EXPLANATION	22
	6.3 HIGH-LEVEL DESCRIPTION OF THE ALGORITHM	23
7	JAVA	25
	7.1 HISTORY	25
	7.1.1 PRIMARY GOALS	26
	7.1.2 THE JAVA PROGRAMMING LANGUAGE	26
	7.1.3 THE JAVA PLATFORM	28
	7.1.4 JAVA RUNTIME ENVIRONMENT	29
	7.2 USES OF JAVA	29
	7.3 JSP (JAVA SERVER PAGE)	30
	7.3.1 THE ADVANTAGES OF JSP	31
8	SERVLETS-FRONT END	33
	8.1 SEVLETS FRONT-END	33
	8.1.1 THE SERVLET RUN-TIME ENVIRONMENT	35
	8.1.2 LIFE CYCLE OF SERVLET	35

	8.2 SERVLET CONFIG AND SERVLET	36
	CONTEXT	
	8.3 JDBC (JAVA DATABASE CONNECTIVITY)	37
	8.4 TESTING	39
9	CONCLUSION	55
10	FUTURE WORK	56
11	REFERENCES	56
12	APPENDIX	57
	12.1 SOURCE CODE	57
	12.2 SCREEN SHOTS	62

LIST OF FIGURES

FIGURE NO:	FIGURE NAME	PAGE NO
4.1.1	DATA FLOW DIAGRAM 0	8
4.1.2	DATA FLOW DIAGRAM 1	9
4.1.3	DATA FLOW DIAGRAM 2	10
4.1.4	DATA FLOW DIAGRAM 3	11
4.1.5	USE CASE DIAGRAM	12
4.1.6	CLASS DIAGRAM	13
4.1.7	SEQUENCE DIAGRAM	14
4.1.8	COLLABORATION DIAGRAM	15
5.1	PUZZLE LOGIN DIAGRAM	17
5.6	ACTIVITY DIAGRAM	19

5.7	ENTITY RELATIONSHIP DIAGRAM	20
7.1.3	AN OVERVIEW OF THE SOFTWARE DEVELOPMENT PROCESS	27
7.3.1	JSP ARCHITECTURE 1	31
7.3.2	JSP ARCHITECTURE 2	33
8.3	JAVA DATABASE CONNECTIVITY DIAGRAM	39

CHAPTER 1

1.OBJECTIVES

OBJECTIVE:

To design and development CaPGP to address a number of security problems altogether, such as online guessing attacks, relay attacks. It offers reasonable security and usability and appears to fit well with some practical applications for improving online security.

1.1 SYNOPSIS:

Many security primitives are based on hard mathematical problems. Using hard AI problems for security is emerging as an exciting new paradigm, but has been underexplored. In this paper, we present a new security primitive based on hard AI problems, namely, a novel family of graphical password systems built on top of Puzzle technology, which we call Puzzle as graphical passwords (CaPRP). CaPRP is both a Puzzle and a graphical password scheme. CaPRP addresses a number of security problems altogether, such as online guessing attacks, relay attacks, and, if combined with dual-view technologies, shoulder-surfing attacks. Notably, a CaPRP password can be found only probabilistically by automatic online guessing attacks even if the password is in the search set. CaPRP also offers a novel approach to address the well-known image hotspot problem in popular graphical password systems, such as Pass Points, that often leads to weak password choices. CaPRP is not a panacea, but it offers reasonable security and usability and appears to fit well with some practical applications for improving online security.

1.2 EXISTING SYSTEM:

Security primitives are based on hard mathematical problems. Using hard AI problems for security is emerging as an exciting new paradigm, but has been under explored. A fundamental task in security is to create cryptographic primitives based on hard mathematical problems that are computationally intractable.

1.3 DISADVANTAGES OF EXISTING SYSTEM:

- This paradigm has achieved just a limited success as compared with the cryptographic primitives based on hard math problems and their wide applications.
- Using hard AI (Artificial Intelligence) problems for security, initially proposed is an exciting new paradigm. Under this paradigm, the most notable primitive invented is Puzzle, which distinguishes human users from computers by presenting a challenge.

CHAPTER 2

2. PROPOSED SYSTEM

PROPOSED SYSTEM:

We present a new security primitive based on hard AI problems, namely, a novel family of graphical password systems built on top of Puzzle technology, which we call Puzzle as graphical passwords (CaPRP). CaPRP is both a Puzzle and a graphical password scheme. CaPRP addresses a number of security problems altogether, such as online guessing attacks, relay attacks, and, if combined with dual-view technologies, shoulder-surfing attacks. Notably, a CaPRP password can be found only probabilistically by automatic online guessing attacks even if the password is in the search set. CaPRP also offers a novel approach to address the well-known image hotspot problem in popular graphical password systems, such as PassPoints, that often leads to weak password choices. CaPRP is not a panacea, but it offers reasonable security and usability and appears to fit well with some practical applications for improving online security. We present exemplary CaPRPs built on both text Puzzle and image-recognition Puzzle. One of them is a text CaPRP wherein a password is a sequence of characters like a text password, but entered by clicking the right character sequence on CaPRP images. CaPRP offers protection against online dictionary attacks on passwords, which have been for long time a major security threat for various online services. This threat is widespread and

considered as a top cyber security risk. Defense against online dictionary attacks is a more subtle problem than it might appear.

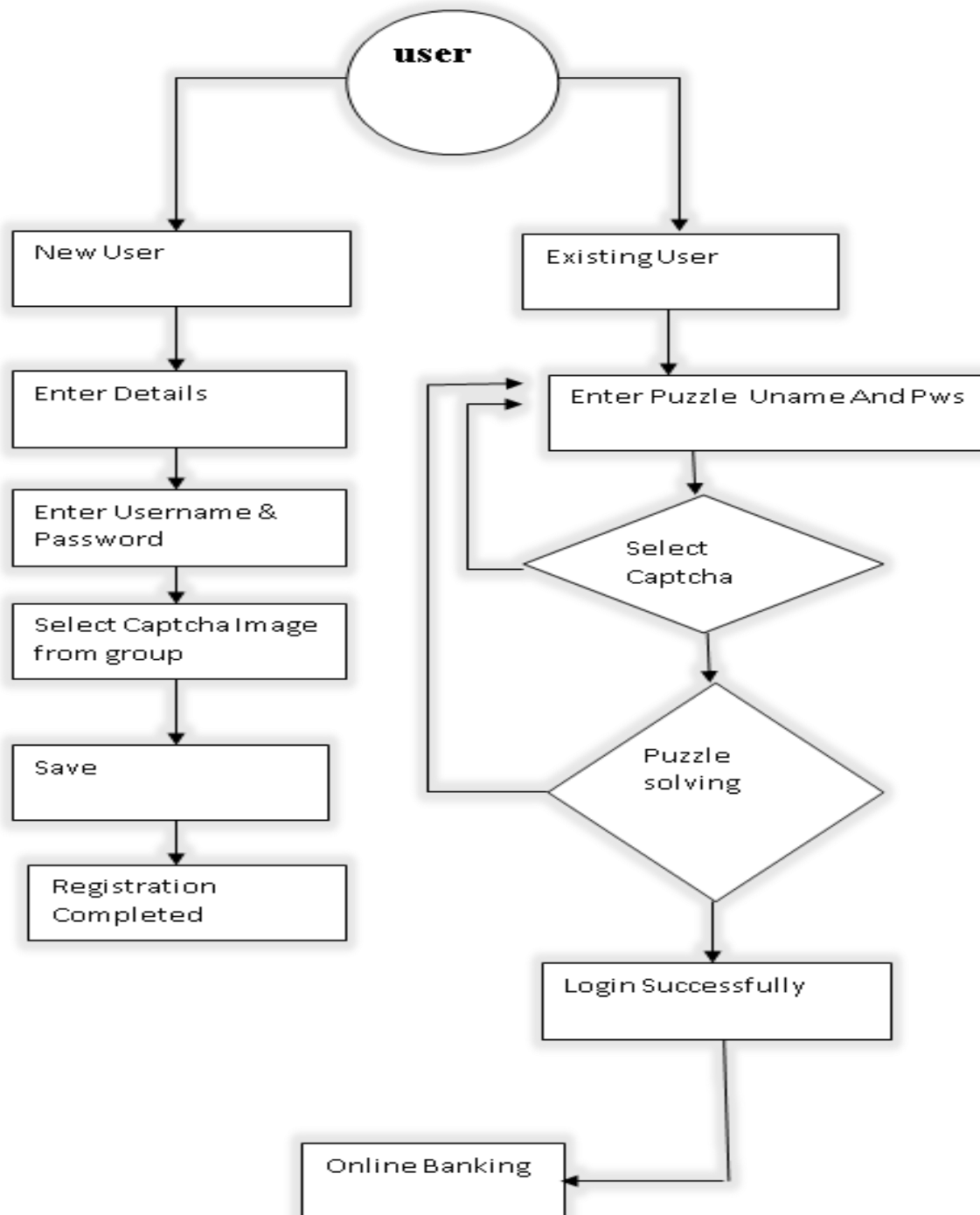
2.1 ADVANTAGES OF PROPOSED SYSTEM:

- It offers reasonable security and usability and appears to fit well with some practical applications for improving online security.
- This threat is widespread and considered as a top cyber security risk. Defense against online dictionary attacks is a more subtle problem than it might appear.
- Puzzle Login (top of Puzzle technology Using mathematical problems).
- Image Puzzle Solving Using AES Algorithm.

CHAPTER 3

3.SYSTEM ARCHITECTURE

SYSTEM ARCHITECTURE:



HARDWARE AND SOFTWARE SPECIFICATION:

3.1 HARDWARE:

1. 1 GB RAM
2. 80 GB Hard Disk
3. Above 2GHz Processor
4. Windows OS

3.2 SOFTWARE:

- ☐ Windows 7 OS
- ☐ JDK 1.7
- ☐ MySQL
- ☐ NetBeans IDE 7.1.2

CHAPTER 4

4.LITERATURE SURVEY

1. GRAPHICAL PASSWORDS: LEARNING FROM THE FIRST TWELVE YEARS

ROBERT BIDDLE, SONIA CHIASSE

We present a new CAPTCHA which is based on identifying an image's up right orientation. This task requires analysis of the often complex contents of an image, a task which humans usually perform well and machines generally do not. Given a large repository of images, such as those from a web search result, we use a suite of automated orientation detectors to prune those images that can be automatically set upright easily. We then apply a social feedback mechanism to verify that the remaining images have a human-recognizable upright orientation.

2. DISTORTION ESTIMATION TECHNIQUES IN SOLVING VISUAL CAPTCHAS

GABRIEL MOY, NATHAN JONES

CAPTCHAs, which are automated tests intended to distinguish humans from programs, are used on many web sites to prevent bot-based account creation and Spam. To avoid imposing undue user friction, CAPTCHAs must be easy for humans and difficult for machines. However, the scientific basis for successful CAPTCHA design is still emerging. This paper examines the widely used class of audio CAPTCHAs based on distorting non-continuous speech with certain classes of noise and demonstrates that virtually all current schemes, including ones from Microsoft, Yahoo, and eBay, are easily broken. More generally, we describe a set of fundamental techniques, packaged together in our De captcha system , that effectively defeat a wide class of audio CAPTCHAs based on non-continuous speech. De captcha's performance on actual observed and synthetic CAPTCHAs indicates that such speech CAPTCHAs are inherently weak and, because of the importance of audio for various classes of users, alternative audio CAPTCHAs must be developed.

3. A NEW GRAPHICAL PASSWORD SCHEME AGAINST SPYWARE BY USING CAPTCHA HAICHANG GAO

CAPTCHAs protect online resources and services from automated access. From an attacker's point of view, they are typically perceived as an annoyance that prevents the mass creation of accounts or the automated posting of messages. Hence, miscreants strive to effectively bypass these protection mechanisms, using techniques such as optical character recognition or machine learning. However, as CAPTCHA systems evolve, they become more resilient against automated analysis approaches. In this paper, we introduce and evaluate an attack that we denote as CAPTCHA smuggling. To perform CAPTCHA smuggling, the attacker slips CAPTCHA challenges into the web browsing sessions of unsuspecting victims, misusing their ability to solve these challenges. A key point of our attack is that the CAPTCHAs are surreptitiously injected into interactions with benign web applications (such as web mail or social networking sites). As a result, they are perceived as a normal part of the application and raise no suspicion. Our evaluation, based on realistic user experiments, shows that CAPTCHA smuggling attacks are feasible in practice.

4. MODELING USER CHOICE IN THE PASSPOINTS GRAPHICAL

PASSWORD SCHEME

AHMET EMIR DIRIK

CAPTCHAs are tests that distinguish humans from software robots in an online environment [3, 14, 7]. We propose and implement three CAPTCHAs based on naming images, distinguishing images, and identifying an anomalous image out of a set. Novel contributions include proposals for two new CAPTCHAs, the user study on image recognition CAPTCHAs, and a new metric for evaluating CAPTCHAs.

5. PAUL C. VAN OORSCHOT

Automated Turing Tests (ATTs), also known as human-in-the-loop techniques, were recently employed in a login protocol by Pinkas and Sander (2002) to protect against online password-guessing attacks. We present modifications providing a new history-based login protocol with ATTs, which uses failed-login counts. Analysis indicates that the new protocol offers opportunities for improved security and user friendliness (fewer ATTs to legitimate users) and greater flexibility (e.g., allowing protocol parameter customization for particular situations and users). We also note that the Pinkas–Sander and other protocols involving ATTs are susceptible to minor variations of well-known middle-person attacks. We discuss complementary techniques to address such attacks, and to augment the security of the original protocol.

4.1 DATA FLOW DIAGRAMS:

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out to precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way. As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple

symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

4.1.1 DATA FLOW DIAGRAM 0:

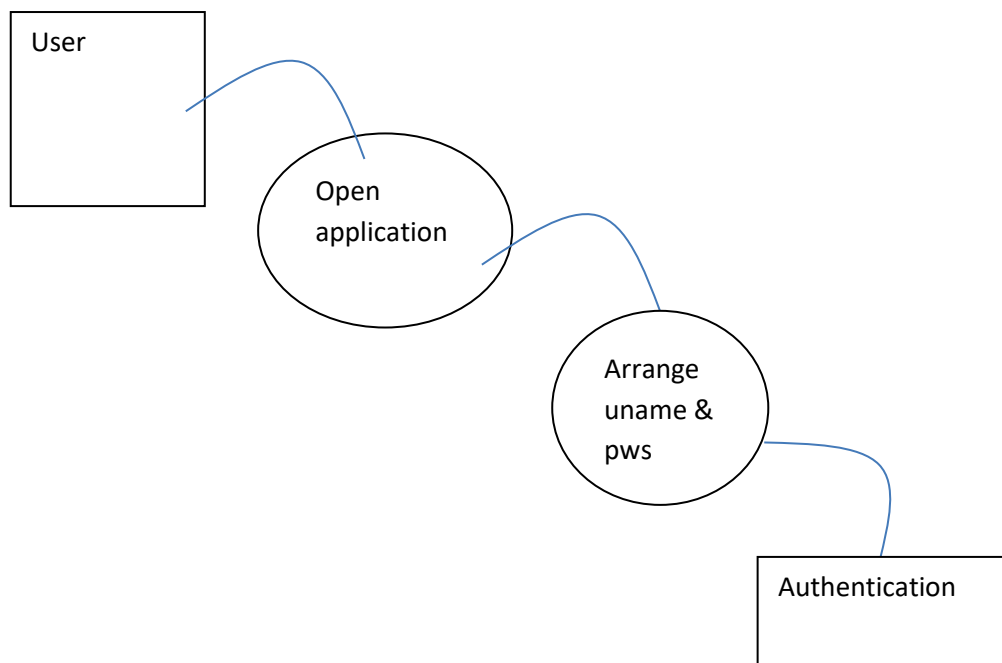


Fig 4.1.1: Data flow diagram 0

4.1.2 DATA FLOW DIAGRAM 1:

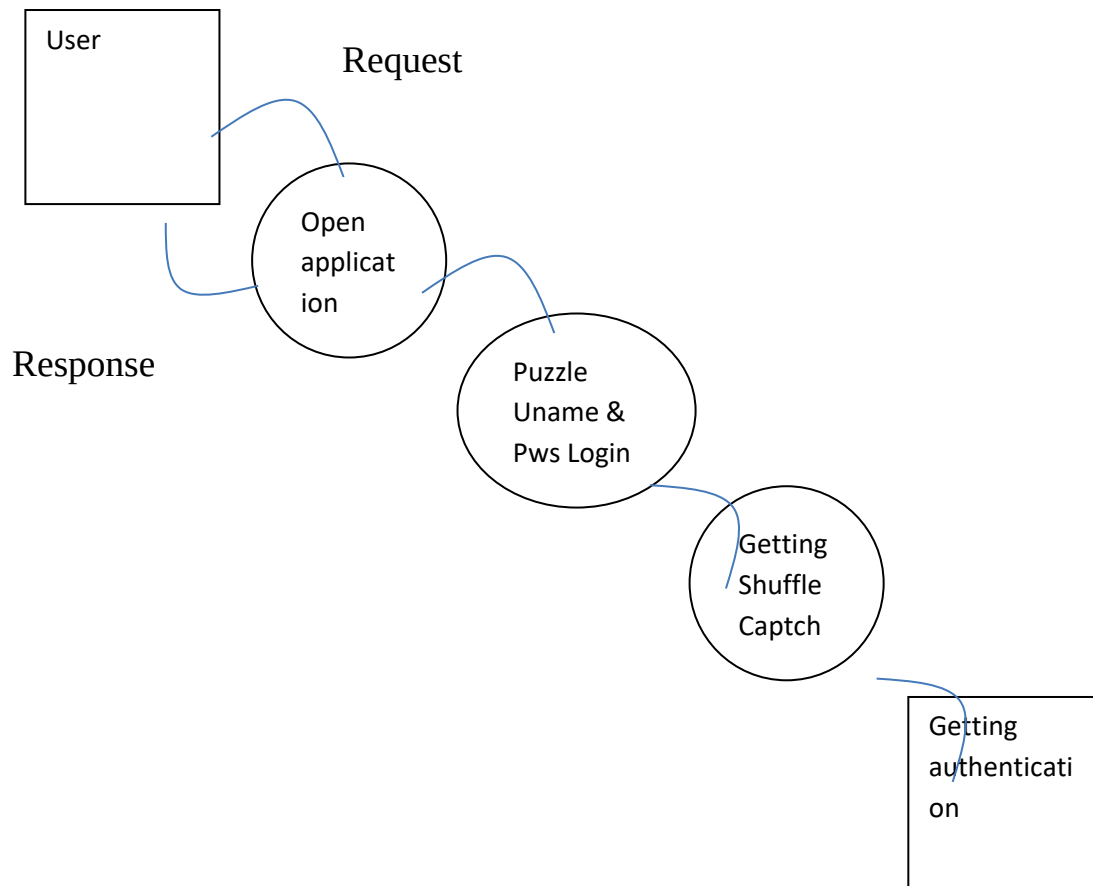


Fig 4.1.2: Data flow diagram 1

4.1.3 DATA FLOW DIAGRAM 2:

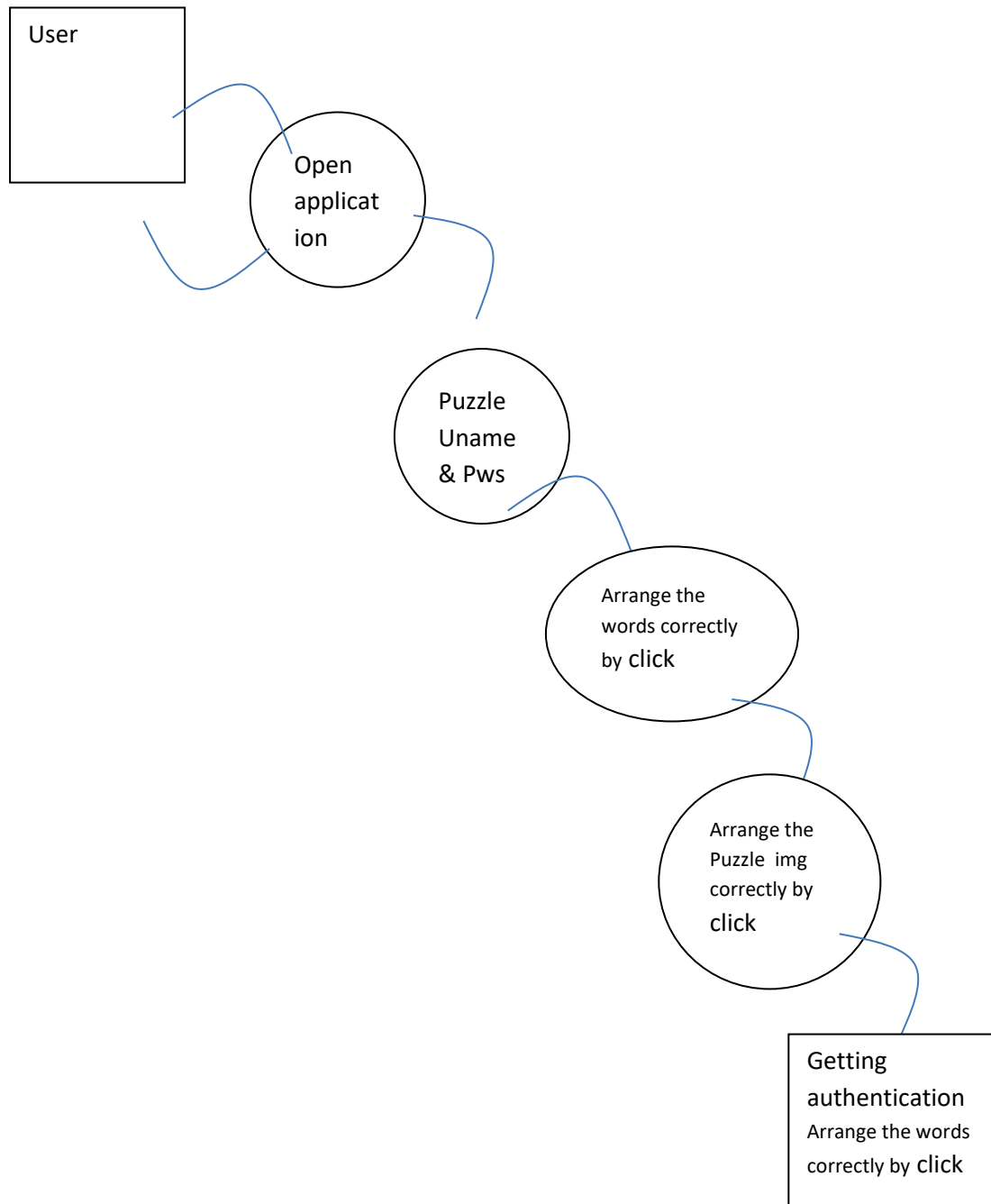


Fig 4.1.3: Data flow diagram 2

4.1.4 DATA FLOW DIAGRAM 3:

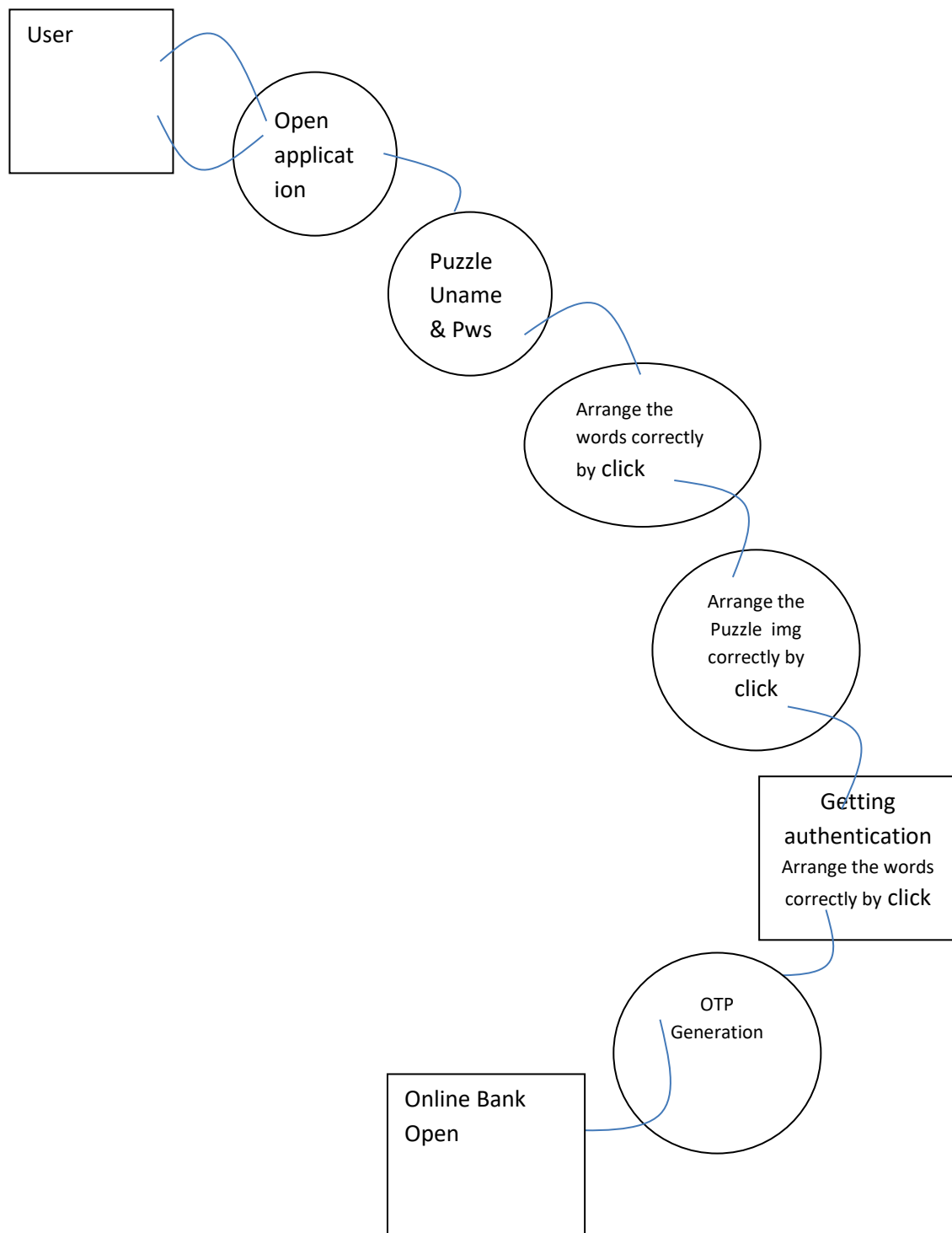


Fig 4.1.4: Data flow diagram 3

4.1.5 USECASE DIAGRAM:

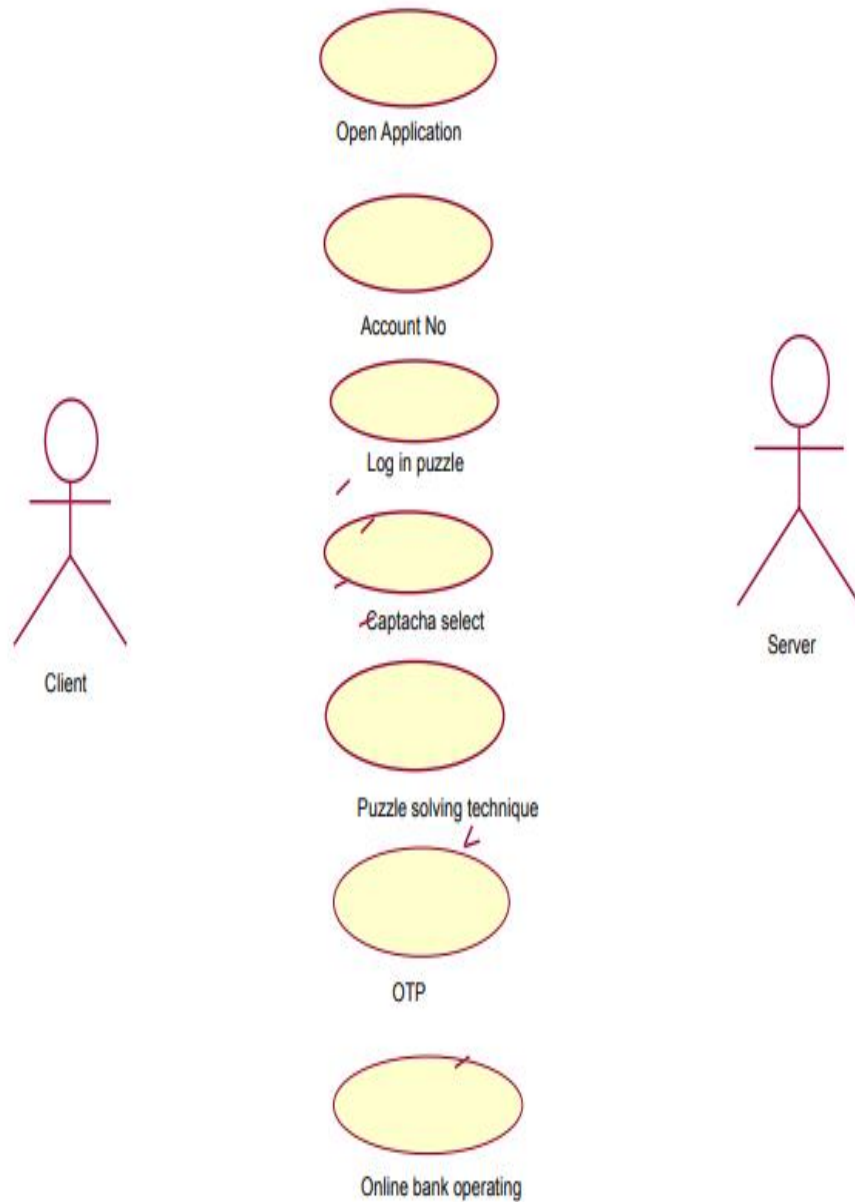


Fig 4.1.5: Use case diagram

Description:

Use case diagrams gives a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted. Here the sender and receiver are the actors and select video, select file, encryption key, decrypt data, extract data and view data are the functions.

4.1.6 CLASS DIAGRAM:-

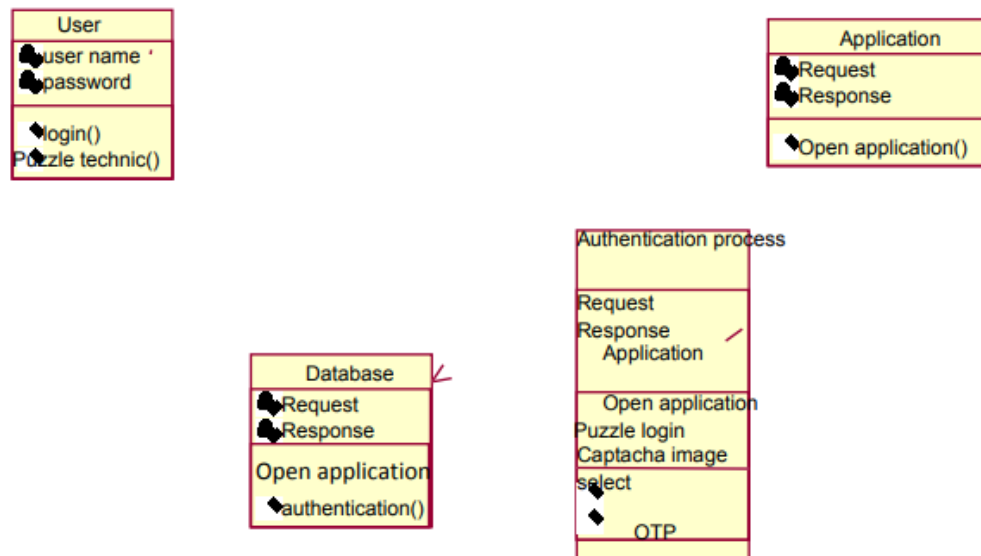


Fig 4.1.6: Class diagram

Description:

It is the main building block of any object oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between each class. A class has three parts, name at the top, attributes in the middle and operations or methods at the bottom. In large systems with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows. Here sender, embed data, receiver, encrypt and decrypt are the classes, each class contains its own attribute and functions, they are related by arrows.

4.1.7 SEQUENCE DIAGRAM:

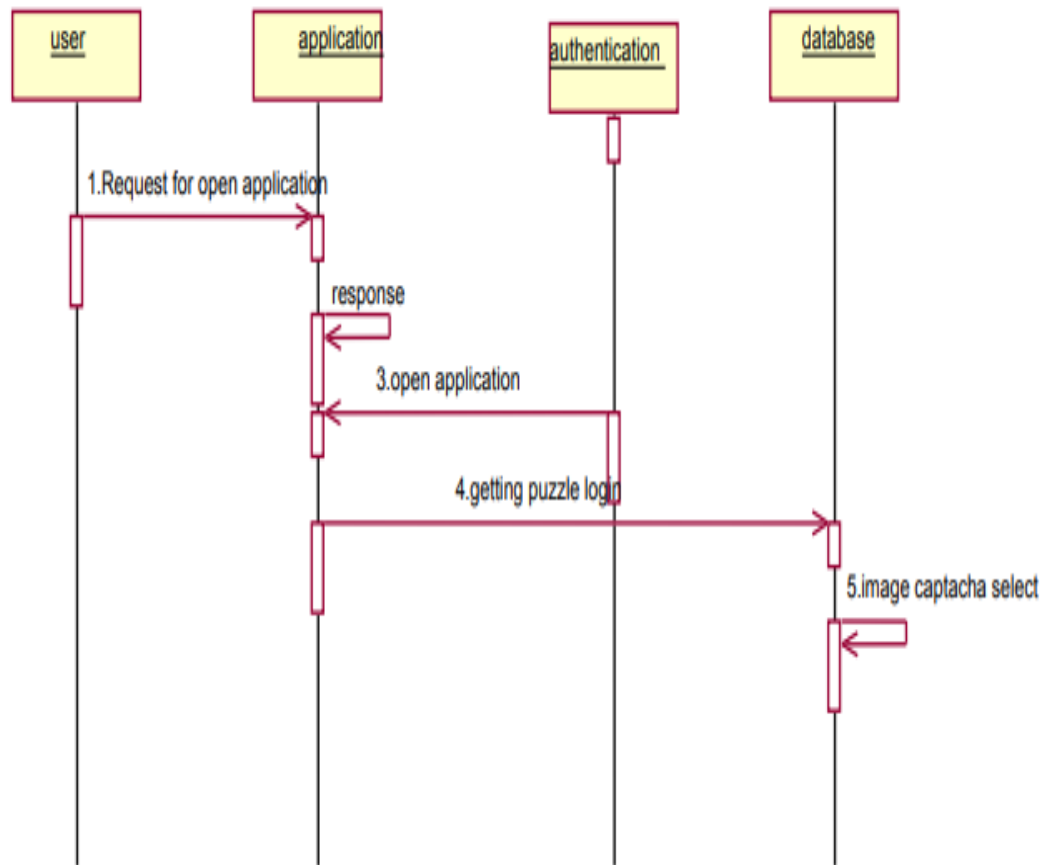


Fig 4.1.7: Sequence diagram

Description:

Sequence diagrams in UML shows how object interact with each other and the order those interactions occur. It's important to note that they show the interactions for a

particular scenario. The processes are represented vertically and interactions are shown as arrows.

Here sender, receiver, embedding data and room are objects they interact each other. The arrow shows interaction like send cover video and data, reserving room etc.

4.1.8 COLLABORATION DIAGRAM:

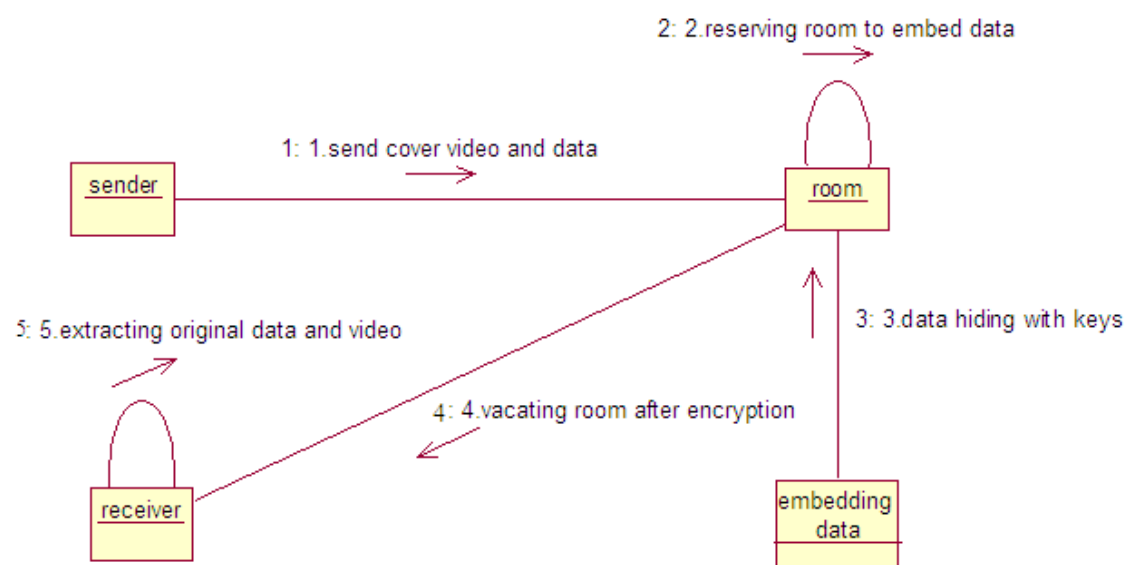


Fig 4.1.8: Collaboration diagram

Description:

Collaboration diagram, is an interaction diagram that shows similar information to sequence diagrams but its primary focus is on object relationships.

On collaboration diagrams, objects are shown with association connectors between them. Messages are added to the associations and show as short arrows pointing in the direction of the message flow. The sequence of messages is shown through a numbering scheme.

Here the objects like sender, receiver, room and Embed data are connected each other and messages are added to the associations.

CHAPTER 5

5.MODULES

MODULES:

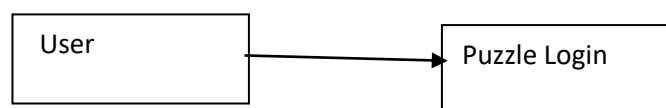
1. PUZZLE LOGIN
2. RANDOM CAPTCHA SELECTION
3. IMAGE PUZZLE SOLVING
4. OTP GENERATION
5. ONLINE BANK

MODULE EXPLANATIONS:

5.1 PUZZLE LOGIN

The security and usability problems in text-based Login and password schemes have resulted in the development of Puzzle password schemes as a possible alternative.

We can visualize the sum $1+2+3+\dots+n$ as a triangle of character . Numbers which have such a pattern of character are called Triangle (or triangular) numbers, written $T(n)$, the sum of the integers from 1 to n time Using Factorial base Login Puzzle Solving.



n	1	2	3	4	5	6
T(n) as a sum	1	1+2	1+2+3	1+2+3+4	1..5	1..6
T(n) as a triangle					---	
T(n)=	1	3	6	10	15	21

Fig 5.1: Puzzle login diagram

5.2 RANDOM CAPTCHA SELECTION

A CAPTCHA is a test that is used to separate humans and machines. CAPTCHA stands for "Completely Automated during test to tell Computers and Humans Apart." It is normally an image test or a simple mathematics problem which a human can read or solve, but a computer cannot. It is made to stop computer [hackers](#) from using a program to automatically set up hundreds of accounts, such as email accounts. It is named after [mathematician](#).

Each individual is chosen [randomly](#) and entirely by chance, such that each individual has the same [probability](#) of being chosen at any stage during the sampling process, and each subset of n individuals has the same probability of being chosen for the sample as any other subset of n individuals This process and technique is known as simple random sampling, and should not be confused with [systematic random sampling](#). A simple random sample is an unbiased surveying technique.

5.3 IMAGE PUZZLE SOLVING

We study how to prevent DoS/DDoS attackers from inflating their puzzle-solving capabilities. To this end, we introduce a new client puzzle referred to as software puzzle. Unlike the existing client puzzle schemes, which publish their puzzle algorithms in advance, a puzzle algorithm in the present software puzzle scheme is randomly generated only after a client request is received at the server side and the algorithm is generated such that: 1) an attacker is unable to prepare an

implementation to solve the puzzle in advance and 2) the attacker needs considerable effort in translating a central processing unit puzzle software to its functionally equivalent GPU version such that the translation cannot be done in real time. Moreover, we show how to implement software puzzle in the generic server-browser model.

5.4 OTP GENERATION

A one-time password (OTP) is a password that is valid for only one login session or transaction, on a [computer](#) system or other digital device. OTPs avoid a number of shortcomings that are associated with traditional (static) [password-based authentication](#); a number of implementations also incorporate [two factor authentication](#) by ensuring that the one-time password requires access to something a person has (such as a small [key ring](#) fob device with the OTP calculator built into it, or a [smartcard](#) or specific [cell phone](#)) as well as something a person knows (such as a PIN).

5.5 ONLINE BANK

Online banking also known as internet banking, e-banking, or virtual banking, is an [electronic payment system](#) that enables customers of a [bank](#) or other [financial institution](#) to conduct a range of [financial transactions](#) through the financial institution's website. The online banking system will typically connect to or be part of the [core banking](#) system operated by a bank and is in contrast to [branch banking](#) that was the traditional way customers access banking services.

5.6 ACTIVITY DIAGRAM:

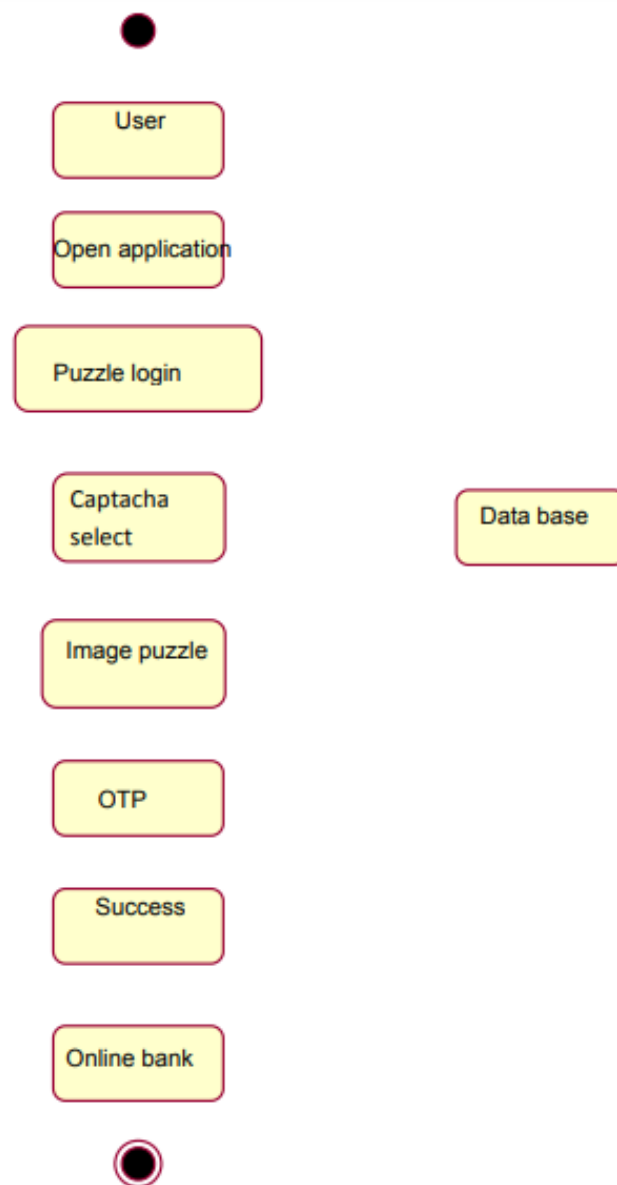


Fig 5.6: Activity diagram

Description:

Activity diagrams represent workflows in an graphical way. They can be used to describe business workflow or the operational workflow of any component in a system

- *rounded rectangles* represent *actions*;

- *diamonds* represent *decisions*;
- *bars* represent the start (*split*) or end (*join*) of concurrent activities;
- a *black circle* represents the start (*initial state*) of the workflow;
- An *encircled black circle* represents the end (*final state*).

5.7 ENTITY RELATIONSHIP (ER) DIAGRAM:

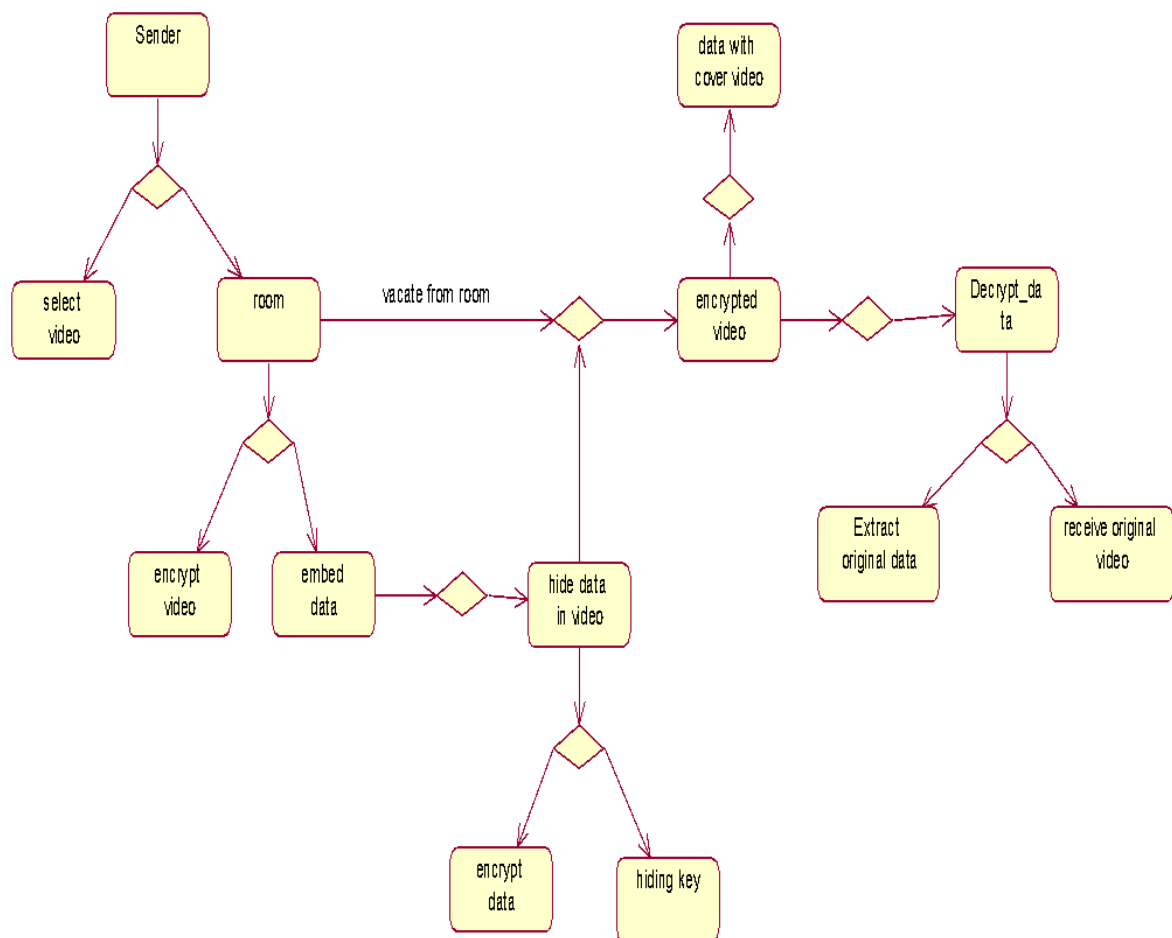


Fig 5.7: Entity Relationship (ER) diagram

Description:

An Entity Relationship Diagram (ERD) is a visual representation of different data using conventions that describe how these data are related to each other.

An entity can be a person, place, event, or object that is relevant to a given system. Here the sender, receiver, etc. are the entities.

A relationship describes how entities interact. Here the sender select video and reserve room so select is a relation.

CHAPTER 6

6.ALGORITHM

ADVANCED ENCRYPTION STANDARD (AES)

6.1 DEFINITION

The Advanced Encryption Standard (AES) is an encryption algorithm for securing sensitive but unclassified material by U.S. Government agencies and, as a likely consequence, may eventually become the de facto encryption standard for commercial transactions in the private sector. (Encryption for the US military and other classified communications is handled by separate, secret algorithms.) In January of 1997, a process was initiated by the National Institute of Standards and Technology (NIST), a unit of the U.S. Commerce Department, to find a more robust replacement for the Data Encryption Standard (DES) and to a lesser degree Triple DES. The specification called for a symmetric algorithm (same key for encryption and decryption) using block encryption (see block cipher) of 128 bits in size, supporting key sizes of 128, 192 and 256 bits, as a minimum. The algorithm was required to be royalty-free for use worldwide and offer security of a sufficient level to protect data for the next 20 to 30 years. It was to be easy to implement in hardware and software, as well as in restricted environments (for example, in a smart card) and offer good defenses against various attack techniques. The entire selection process was fully open to public scrutiny and comment, it being decided that full visibility would ensure the best possible analysis of the designs. In 1998, the NIST selected 15 candidates for the AES, which were then subject to preliminary analysis by the world cryptographic community, including the National Security Agency. On the basis of this, in August 1999, NIST selected five algorithms for more extensive analysis. These were:

- MARS, submitted by a large team from IBM Research
- RC6, submitted by RSA Security

- Rijndael, submitted by two Belgian cryptographers, Joan Daemen and Vincent Rijmen.
- Serpent, submitted by Ross Andersen, Eli Biham and Lars Knudsen
- Twofish, submitted by a large team of researchers including Counterpane's respected cryptographer, Bruce Schneier

Implementations of all of the above were tested extensively in ANSI C and Java languages for speed and reliability in such measures as encryption and decryption speeds, key and algorithm set-up time and resistance to various attacks, both in hardware- and software-centric systems. Once again, detailed analysis was provided by the global cryptographic community (including some teams trying to break their own submissions). The end result was that on October 2, 2000, NIST announced that Rijndael had been selected as the proposed standard. On December 6, 2001, the Secretary of Commerce officially approved Federal Information Processing Standard (FIPS) 197, which specifies that all sensitive, unclassified documents will use Rijndael as the Advanced Encryption Standard. Also see cryptography, data recovery agent (DRA) **RELATED GLOSSARY TERMS:** RSA algorithm (Rivest-Shamir-Adleman), data key, greynet (or graynet), spam cocktail (or anti-spam cocktail), finger scanning (fingerprint scanning), munging, insider threat, authentication server, defense in depth, nonrepudiation.

6.2 EXPLANATIONS:

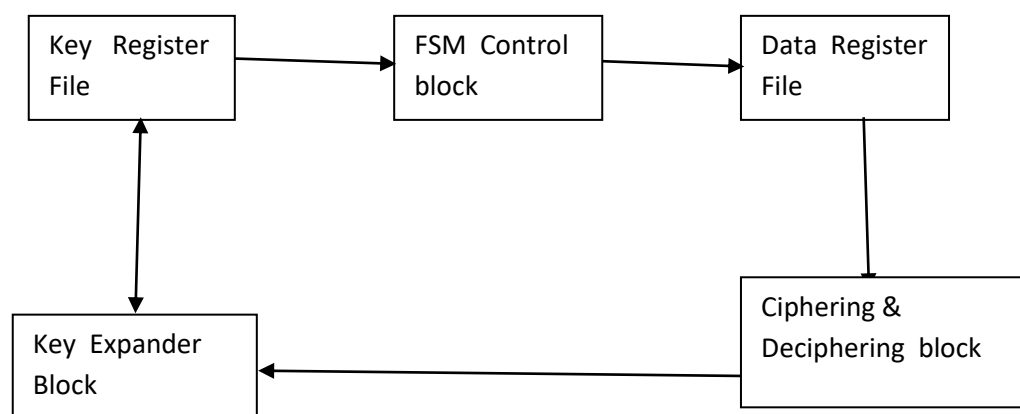
AES is based on a design principle known as a Substitution permutation network. It is fast in both software and hardware. Unlike its predecessor, DES, AES does not use a Feistel network. AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits. The block size has a maximum of 256 bits, but the key size has no theoretical maximum. AES operates on a 4×4 column-major order matrix of bytes, termed the *state* (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field. The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round consists of several processing steps, including one that

depends on the encryption key. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

6.3 HIGH-LEVEL DESCRIPTION OF THE ALGORITHM

1. KeyExpansion—round keys are derived from the cipher key using Rijndael's key schedule
2. Initial Round
 1. AddRoundKey—each byte of the state is combined with the round key using bitwise xor
3. Rounds
 1. SubBytes—a non-linear substitution step where each byte is replaced with another according to lookup.
 2. ShiftRows—a transposition step where each row of the state is shifted cyclically a certain number of steps.
 3. MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
 4. AddRoundKey
4. Final Round (no MixColumns)
 1. SubBytes
 2. ShiftRows
 3. AddRoundKey

Diagrams



Examples

In this appendix, twenty examples are provided for the MAC generation process. The underlying block cipher is either the AES algorithm or TDEA. A block cipher key is fixed for each of the currently allowed key sizes, i.e., AES-128, AES-192, AES-256, two key TDEA, and three key TDEA. For each key, the generation of the associated subkeys is given, followed by four examples of MAC generation with the key. The messages in each set of examples are derived by truncating a common fixed string of 64 bytes. All strings are represented in hexadecimal notation, with a space (or a new line) inserted every 8 symbols, for readability. As in the body of the Recommendation, K_1 and K_2 denote the subkeys, M denotes the message, and T denotes the MAC. For the AES algorithm examples, T_{len} is 128, i.e., 32 hexadecimal symbols, and K denotes the key. For the TDEA examples, T_{len} is 64, i.e., 16 hexadecimal symbols, and the key, K , is the ordered triple of strings, (Key_1 , Key_2 , Key_3). For two key TDEA, $Key_1 = Key_3$.

D.1 AES-128

For Examples 1–4 below, the block cipher is the AES algorithm with the following 128 bit key:

K 2b7e1516 28aed2a6 abf71588 09cf4f3c.

Subkey Generation

$CIPH(K(0$

128

) 7df76b0c 1ab899b3 3e42f047 b91b546f

K_1 fbeed618 35713366 7c85e08f 7236a8de

K_2 f7ddac30 6ae266cc f90bc11e e46d513b

Example Explanations

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a

symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

CHAPTER 7

7.JAVA

JAVA (PROGRAMMING LANGUAGE)

7.1 HISTORY

The JAVA language was created by James Gosling in June 1991 for use in a set top box project. The language was initially called Oak, after an oak tree that stood outside Gosling's office - and also went by the name Green - and ended up later being renamed to Java, from a list of random words. Gosling's goals were to implement a virtual machine and a language that had a familiar C/C++ style of notation. The first public implementation was Java 1.0 in 1995. It promised "Write Once, Run Anywhere" (WORA), providing no-cost runtimes on popular platforms. It was fairly secure and its security was configurable, allowing network and file access to be restricted. Major web browsers soon incorporated the ability to run secure Java applets within web pages. Java quickly became popular. With the advent of Java 2, new versions had multiple configurations built for different types of platforms. For example, J2EE was for enterprise applications and the greatly stripped down version J2ME was for mobile applications. J2SE was the designation for the Standard Edition. In 2006, for marketing purposes, new J2 versions were renamed Java EE, Java ME, and Java SE, respectively.

In 1997, Sun Microsystems approached the ISO/IEC JTC1 standards body and later the Ecma International to formalize Java, but it soon withdrew from the process. Java remains a standard that is controlled through the Java Community Process. At one time, Sun made most of its Java implementations available without charge although they were proprietary software. Sun's revenue from Java was generated by the selling of licenses for specialized products such as the Java

Enterprise System. Sun distinguishes between its Software Development Kit (SDK) and Runtime Environment (JRE) which is a subset of the SDK, the primary distinction being that in the JRE, the compiler, utility programs, and many necessary header files are not present.

On 13 November 2006, Sun released much of Java as free software under the terms of the GNU General Public License (GPL). On 8 May 2007 Sun finished the process, making all of Java's core code open source, aside from a small portion of code to which Sun did not hold the copyright.

7.1.1 PRIMARY GOALS

There were five primary goals in the creation of the Java language:

- It should use the object-oriented programming methodology.
- It should allow the same program to be executed on multiple operating systems.
- It should contain built-in support for using computer networks.
- It should be designed to execute code from remote sources securely.
- It should be easy to use by selecting what were considered the good parts of other object-oriented languages

7.1.2 THE JAVA PROGRAMMING LANGUAGE:

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance

Each of the preceding buzzwords is explained in *The Java Language Environment* , a white paper written by James Gosling and Henry McGilton.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler.

A .class file does not contain code that is native to your processor; it instead contains *byte codes* — the machine language of the Java Virtual Machine¹ (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

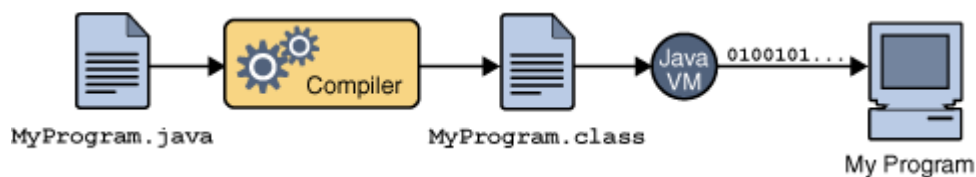
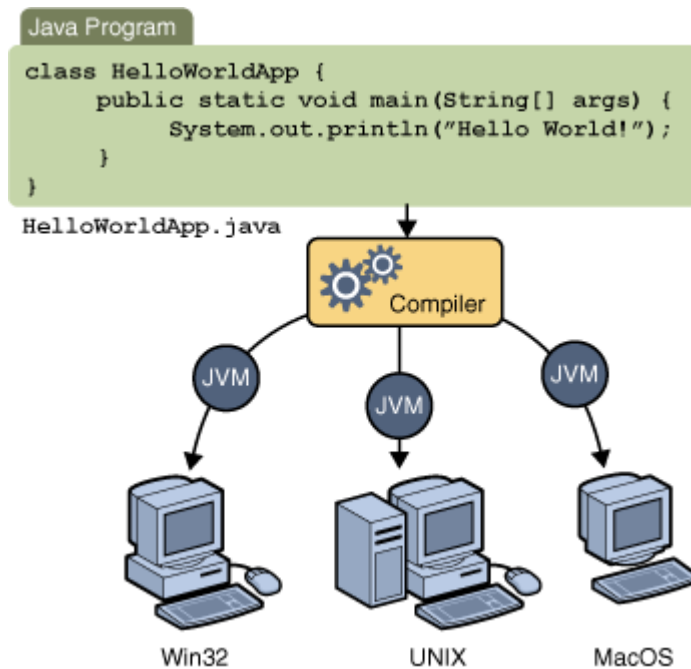


Fig 7.1.3: An overview of the software development process.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the SolarisTM Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the Java Hot Spot virtual machine perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.



Through the Java VM, the same application is capable of running on multiple platforms.

7.1.3 THE JAVA PLATFORM

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

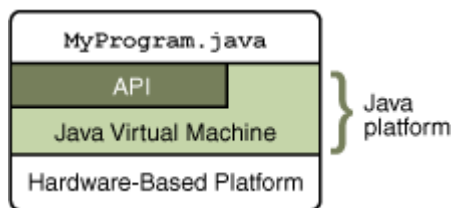
The Java Virtual Machine

The Java Application Programming Interface (API)

You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces;

these libraries are known as *packages*. The next section, What Can Java Technology Do? Highlights some of the functionality provided by the API.



The API and Java Virtual Machine insulate the program from the underlying hardware.

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

7.1.4 JAVA RUNTIME ENVIRONMENT

The Java Runtime Environment, or JRE, is the software required to run any application deployed on the Java Platform. End-users commonly use a JRE in software packages and Web browser plug-in. Sun also distributes a superset of the JRE called the Java 2 SDK (more commonly known as the JDK), which includes development tools such as the Javac compiler, Javadoc, Jar and debugger.

One of the unique advantages of the concept of a runtime engine is that errors (exceptions) should not 'crash' the system. Moreover, in runtime engine environments such as Java there exist tools that attach to the runtime engine and every time that an exception of interest occurs they record debugging information that existed in memory at the time the exception was thrown (stack and heap values). These Automated Exception Handling tools provide 'root-cause' information for exceptions in Java programs that run in production, testing or development environments.

7.2 USES OF JAVA

Blue is a smart card enabled with the secure, cross-platform, object-oriented Java Card API and technology. Blue contains an actual on-card processing chip,

allowing for enhance able and multiple functionality within a single card. Applets that comply with the Java Card API specification can run on any third-party vendor card that provides the necessary Java Card Application Environment (JCAE). Not only can multiple applet programs run on a single card, but new applets and functionality can be added after the card is issued to the customer

- Java Can be used in Chemistry.
- In NASA also Java is used.
- In 2D and 3D applications java is used.
- In Graphics Programming also Java is used.
- In Animations Java is used.
- In Online and Web Applications Java is used.

7.3 JAVA SERVER PAGE :

Java Server Pages (JSP) is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request. The technology allows Java code and certain pre-defined actions to be embedded into static content.

The JSP syntax adds additional XML-like tags, called JSP actions, to be used to invoke built-in functionality. Additionally, the technology allows for the creation of JSP tag libraries that act as extensions to the standard HTML or XML tags. Tag libraries provide a platform independent way of extending the capabilities of a Web server.

JSPs are compiled into Java Servlet by a JSP compiler. A JSP compiler may generate a servlet in Java code that is then compiled by the Java compiler, or it may generate byte code for the servlet directly. JSPs can also be interpreted on-the-fly reducing the time taken to reload changes

JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic web content. JSP technology enables rapid development of web-based applications that are server and platform-independent.

7.3.1 ARCHITECTURE OF JSP 1

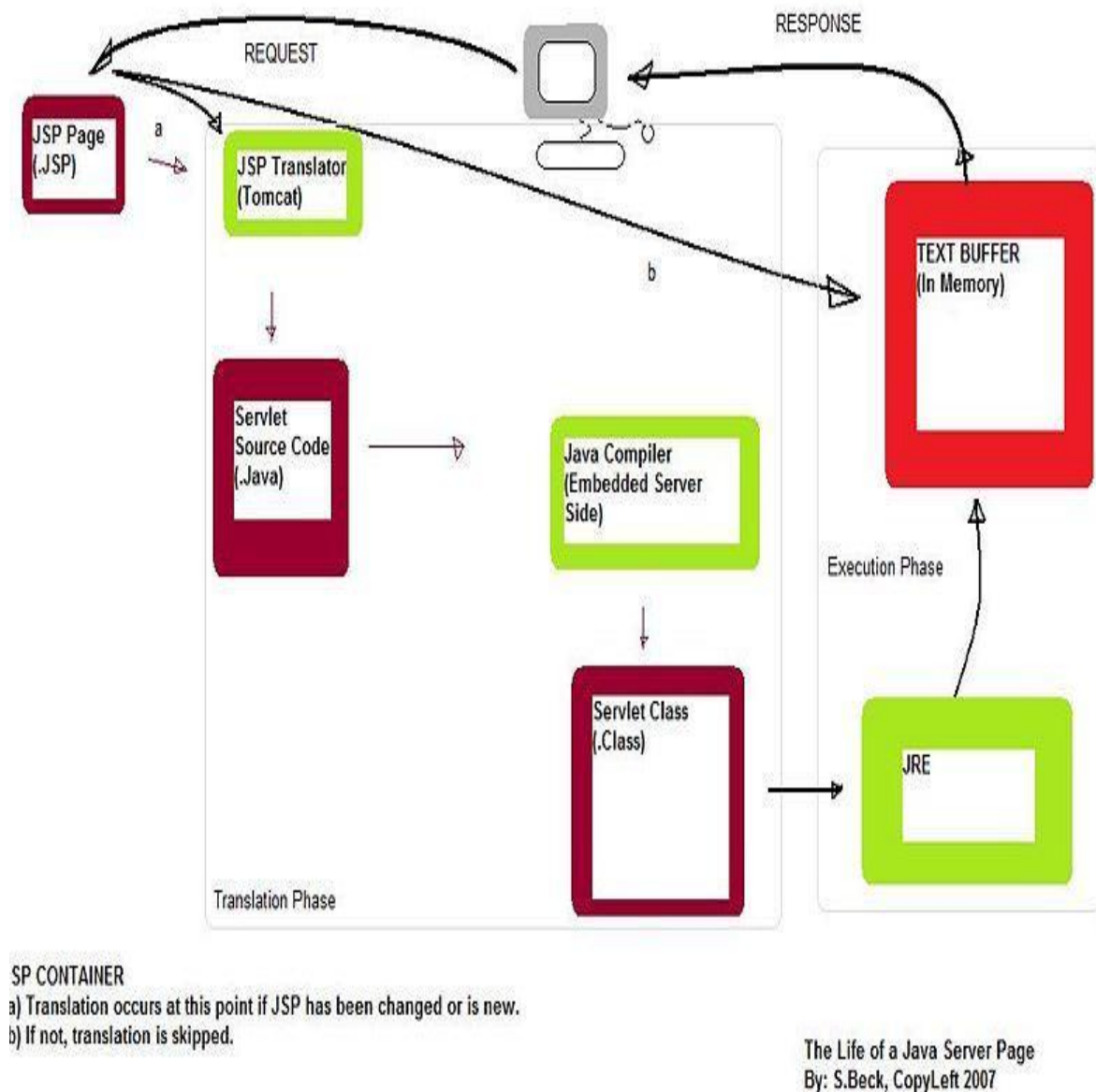


Fig 7.3.1: Architecture of JSP 1

7.3.1 THE ADVANTAGES OF JSP

Active Server Pages (ASP). ASP is a similar technology from Microsoft. The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS-specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers. Pure Servlet. JSP doesn't give you anything that you couldn't in principle do with a

Servlet. But it is more convenient to write (and to modify!) regular HTML than to have a zillion `println` statements that generate the HTML. Plus, by separating the look from the content you can put different people on different tasks: your Web page design experts can build the HTML, leaving places for your Servlet programmers to insert the dynamic content.

Server-Side Includes (SSI). SSI is a widely-supported technology for including externally-defined pieces into a static Web page. JSP is better because it lets you use Servlet instead of a separate program to generate that dynamic part. Besides, SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like. JavaScript. JavaScript can generate HTML dynamically on the client. This is a useful capability, but only handles situations where the dynamic information is based on the client's environment.

With the exception of cookies, HTTP and form submission data is not available to JavaScript. And, since it runs on the client, JavaScript can't access server-side resources like databases, catalogs, pricing information, and the like. Static HTML. Regular HTML, of course, cannot contain dynamic information. JSP is so easy and convenient that it is quite feasible to augment HTML pages that only benefit marginally by the insertion of small amounts of dynamic data. Previously, the cost of using dynamic data would preclude its use in all but the most valuable instances.

7.3.2 ARCHITECTURE OF JSP 2

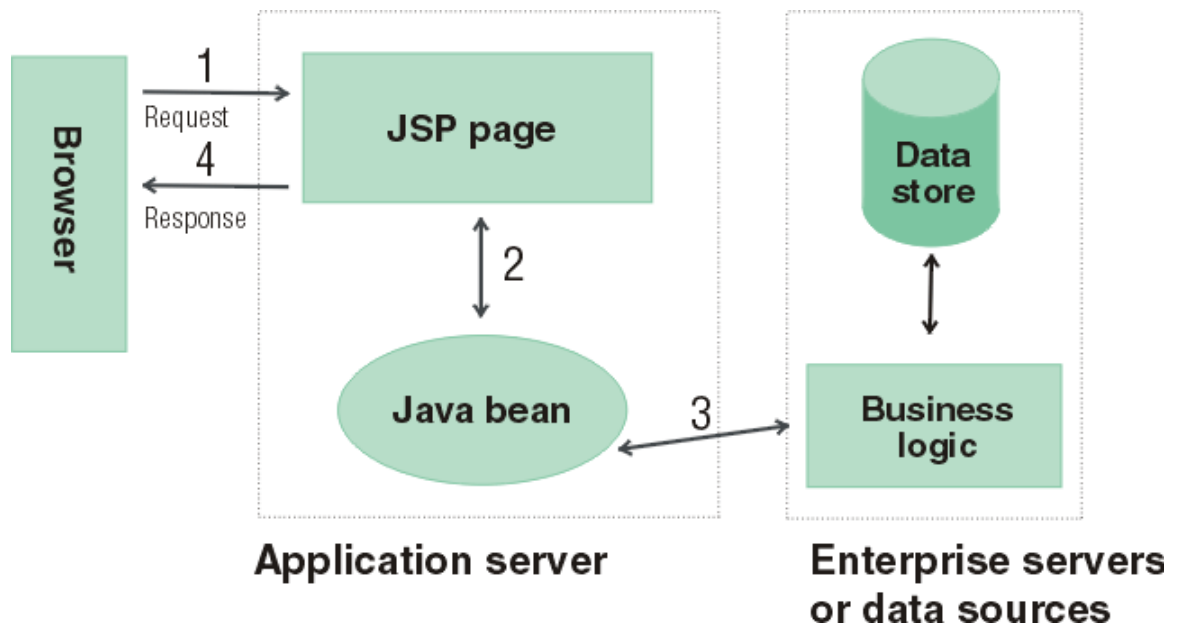


Fig7.3.2: Architecture of JSP 2

- The browser sends a request to a JSP page.
- The JSP page communicates with a Java bean.
- The Java bean is connected to a database.
- The JSP page responds to the browser.

CHAPTER 8

8.SERVLETS-FRONT END

8.1 SERVLETS – FRONT END

The Java Servlet API allows a software developer to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlet are the Java counterpart to non-Java dynamic Web content technologies such as PHP, CGI and ASP.NET. Servlet can

maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting.

The Servlet API, contained in the Java package hierarchy `javax. Servlet`, defines the expected interactions of a Web container and a Servlet. A Web container is essentially the component of a Web server that interacts with the Servlet. The Web container is responsible for managing the lifecycle of Servlet, mapping a URL to a particular Servlet and ensuring that the URL requester has the correct access rights.

A Servlet is an object that receives a request and generates a response based on that request. The basic Servlet package defines Java objects to represent Servlet requests and responses, as well as objects to reflect the Servlet configuration parameters and execution environment. The package `javax. Servlet. Http` defines HTTP-specific subclasses of the generic Servlet elements, including session management objects that track multiple requests and responses between the Web server and a client. Servlet may be packaged in a WAR file as a Web application.

Servlet can be generated automatically by Java Server Pages(JSP), or alternately by template engines such as Web Macro. Often Servlet are used in conjunction with JSPs in a pattern called "Model 2", which is a flavour of the model-view-controller pattern.

Servlet are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages. Building Web pages on the fly is useful (and commonly done) for a number of reasons:.

The Web page is based on data submitted by the user. For example the results pages from search engines are generated this way, and programs that process orders for e-commerce sites do this as well. The data changes frequently. For example, a weather-report or news headlines page might build the page dynamically, perhaps returning a previously built page if it is still up to date. The Web page uses information from corporate databases or other such sources. For

example, you would use this for making a Web page at an on-line store that lists current prices and number of items in stock.

8.1.1 THE SERVLET RUN-TIME ENVIRONMENT

A Servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a Servlet engine. The Servlet engine loads the servlet class the first time the Servlet is requested, or optionally already when the Servlet engine is started. The Servlet then stays loaded to handle multiple requests until it is explicitly unloaded or the Servlet engine is shut down.

Some Web servers, such as Sun's Java Web Server (JWS), W3C's Jigsaw and Gefion Software's Lite Web Server (LWS) are implemented in Java and have a built-in Servlet engine. Other Web servers, such as Netscape's Enterprise Server, Microsoft's Internet Information Server (IIS) and the Apache Group's Apache, require a Servlet engine add-on module. The add-on intercepts all requests for Servlet, executes them and returns the response through the Web server to the client. Examples of Servlet engine add-ons are Gefion Software's WAI Cool Runner, IBM's Web Sphere, Live Software's JRun and New Atlanta's Servlet Exec.

All Servlet API classes and a simple Servlet-enabled Web server are combined into the Java Servlet Development Kit (JSDK), available for download at Sun's official Servlet site .To get started with Servlet I recommend that you download the JSDK and play around with the sample Servlet.

8.1.2 LIFE CYCLE OF SERVLET

- The Servlet lifecycle consists of the following steps:
- The Servlet class is loaded by the container during start-up.

The container calls the `init()` method. This method initializes the Servlet and must be called before the Servlet can service any requests. In the entire life of a Servlet, the `init()` method is called only once. After initialization, the Servlet can service client-requests.

Each request is serviced in its own separate thread. The container calls the `service()` method of the Servlet for every request.

The `service()` method determines the kind of request being made and dispatches it to an appropriate method to handle the request. The developer of the Servlet must provide an implementation for these methods. If a request for a method that is not implemented by the Servlet is made, the method of the parent class is called, typically resulting in an error being returned to the requester. Finally, the container calls the `destroy()` method which takes the Servlet out of service. The `destroy()` method like `init()` is called only once in the lifecycle of a Servlet.

Request and Response Objects

The `doGet` method has two interesting parameters: `HttpServletRequest` and `HttpServletResponse`. These two objects give you full access to all information about the request and let you control the output sent to the client as the response to the request. With CGI you read environment variables and `stdin` to get information about the request, but the names of the environment variables may vary between implementations and some are not provided by all Web servers.

The `HttpServletRequest` object provides the same information as the CGI environment variables, plus more, in a standardized way. It also provides methods for extracting HTTP parameters from the query string or the request body depending on the type of request (GET or POST). As a Servlet developer you access parameters the same way for both types of requests. Other methods give you access to all request headers and help you parse date and cookie headers.

Instead of writing the response to `stdout` as you do with CGI, you get an `OutputStream` or a `PrintWriter` from the `HttpServletResponse`. The `OutputStream` is intended for binary data, such as a GIF or JPEG image, and the `PrintWriter` for text output. You can also set all response headers and the status code, without having to rely on special Web server CGI configurations such as Non Parsed Headers (NPH). This makes your Servlet easier to install.

8.2 SERVLET CONFIG AND SERVLETCONTEXT

There is only one `ServletContext` in every application. This object can be used by all the Servlet to obtain application level information or container details. Every Servlet, on the other hand, gets its own `ServletConfig` object. This object provides

initialization parameters for a servlet. A developer can obtain the reference to `ServletContext` using either the `ServletConfig` object or `ServletRequest` object.

All servlets belong to one servlet context. In implementations of the 1.0 and 2.0 versions of the Servlet API all servlets on one host belongs to the same context, but with the 2.1 version of the API the context becomes more powerful and can be seen as the humble beginnings of an Application concept. Future versions of the API will make this even more pronounced.

Many servlet engines implementing the Servlet 2.1 API let you group a set of servlets into one context and support more than one context on the same host. The `ServletContext` in the 2.1 API is responsible for the state of its servlets and knows about resources and attributes available to the servlets in the context. Here we will only look at how `ServletContext` attributes can be used to share information among a group of servlets.

There are three `ServletContext` methods dealing with context attributes: `getAttribute`, `setAttribute` and `removeAttribute`. In addition the servlet engine may provide ways to configure a servlet context with initial attribute values. This serves as a welcome addition to the servlet initialization arguments for configuration information used by a group of servlets, for instance the database identifier we talked about above, a style sheet URL for an application, the name of a mail server, etc.

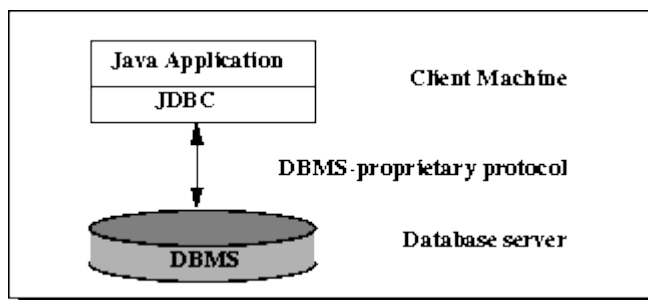
8.3 JDBC (JAVADATABASE CONNECTIVITY)

Java Database Connectivity (JDBC) is a programming framework for Java developers writing programs that access information stored in databases, spread sheets, and flat files. JDBC is commonly used to connect a user program to a "behind the scenes" database, regardless of what database management software is used to control the database. In this way, JDBC is cross-platform . This article will provide an introduction and sample code that demonstrates database access from Java programs that use the classes of the JDBC API, which is available for free download from Sun's site.

A database that another program links to is called a data source. Many data sources, including products produced by Microsoft and Oracle, already use a

standard called Open Database Connectivity (ODBC). Many legacy C and Perl programs use ODBC to connect to data sources. ODBC consolidated much of the commonality between database management systems. JDBC builds on this feature, and increases the level of abstraction. JDBC-ODBC bridges have been created to allow Java programs to connect to ODBC-enabled database software .

The JDBC API supports both two-tier and three-tier processing models for database access.



In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user.

MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of

applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

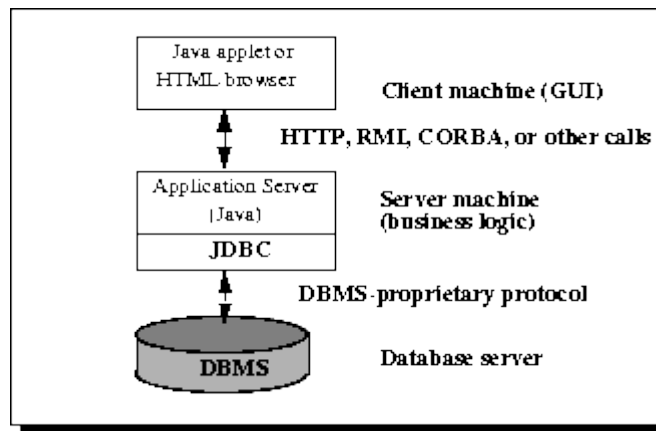


Fig 8.3: Java database connectivity diagram

Until recently, the middle tier has often been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java byte code into efficient machine-specific code and technologies such as Enterprise JavaBeans™, the Java platform is fast becoming the standard platform for middle-tier development. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features.

With enterprises increasingly using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of a three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed transactions, and disconnected row sets. The JDBC API is also what allows access to a data source from a Java middle tier.

8.4 TESTING

The various levels of testing are

1. White Box Testing
2. Black Box Testing
3. Unit Testing
4. Functional Testing

5. Performance Testing
6. Integration Testing
7. Objective
8. Integration Testing
9. Validation Testing
10. System Testing
11. Structure Testing
12. Output Testing
13. User Acceptance Testing

WHITE BOX TESTING

White-box testing (also known as **clear box testing**, **glass box testing**, **transparent box testing**, and **structural testing**) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system-level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

White-box test design techniques include:

- Control flow testing
- Data flow testing

- Branch testing
- Path testing
- Statement coverage
- Decision coverage

White-box testing is a method of testing the application at the level of the source code. The test cases are derived through the use of the design techniques mentioned above: control flow testing, data flow testing, branch testing, path testing, statement coverage and decision coverage as well as modified condition/decision coverage. White-box testing is the use of these techniques as guidelines to create an error free environment by examining any fragile code.

These White-box testing techniques are the building blocks of white-box testing, whose essence is the careful testing of the application at the source code level to prevent any hidden errors later on. These different techniques exercise every visible path of the source code to minimize errors and create an error-free environment. The whole point of white-box testing is the ability to know which line of the code is being executed and being able to identify what the correct output should be.

Levels

1. Unit testing. White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.
2. Integration testing. White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.

3. Regression testing. White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

White-box testing's basic procedures involve the understanding of the source code that you are testing at a deep level to be able to test them. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analysed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

1. Input, involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white-box testing to layout all of the basic information.
2. Processing Unit, involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
3. Output, prepare final report that encompasses all of the above preparations and results.

BLACK BOX TESTING

Black-box testing is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings (see white-box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

TEST PROCEDURES

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of *what* the

software is supposed to do but is not aware of *how* it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of *how* the software produces the output in the first place.

TEST CASES

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily *functional* in nature, *non-functional* tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

TEST DESIGN TECHNIQUES

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition tables
- Equivalence partitioning
- Boundary value analysis

UNIT TESTING

In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are created by programmers or occasionally by white box testers during the development process.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. Its implementation can vary from being very manual (pencil and paper) to being formalized as part of build automation.

Testing will not catch every error in the program, since it cannot evaluate every execution path in any but the most trivial programs. The same is true for unit testing. Additionally, unit testing by definition only tests the functionality of the units themselves. Therefore, it will not catch integration errors or broader system-level errors (such as functions performed across multiple units, or non-functional test areas such as performance).

Unit testing should be done in conjunction with other software testing activities, as they can only show the presence or absence of particular errors; they cannot prove a complete absence of errors. In order to guarantee correct behaviour for every execution path and every possible input, and ensure the absence of errors, other techniques are required, namely the application of formal methods to proving that a software component has no unexpected behaviour.

Software testing is a combinatorial problem. For example, every Boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

This obviously takes time and its investment may not be worth the effort. There are also many problems that cannot easily be tested at all – for example those that are nondeterministic or involve multiple threads. In addition, code for a unit test is likely to be at least as buggy as the code it is testing. Fred Brooks in *The Mythical Man-Month* quotes: *never take two chronometers to sea. Always take one or three*. Meaning, if two chronometers contradict, how do you know which one is correct?

Another challenge related to writing the unit tests is the difficulty of setting up realistic and useful tests. It is necessary to create relevant initial conditions so the part of the application being tested behaves like part of the complete system. If these

initial conditions are not set correctly, the test will not be exercising the code in a realistic context, which diminishes the value and accuracy of unit test results.

To obtain the intended benefits from unit testing, rigorous discipline is needed throughout the software development process. It is essential to keep careful records not only of the tests that have been performed, but also of all changes that have been made to the source code of this or any other unit in the software. Use of a version control system is essential. If a later version of the unit fails a particular test that it had previously passed, the version-control software can provide a list of the source code changes (if any) that have been applied to the unit since that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately if such a process is not implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite, increasing false positives and reducing the effectiveness of the test suite.

Unit testing embedded system software presents a unique challenge: Since the software is being developed on a different platform than the one it will eventually run on, you cannot readily run a test program in the actual deployment environment, as is possible with desktop programs.

FUNCTIONAL TESTING

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional Testing usually describes *what* the system does.

Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate* a program by checking it against the published user or system requirements" (Kane, Falk, Nguyen 1999, p. 52).

Functional testing typically involves five steps .The identification of functions that the software is expected to perform

1. The creation of input data based on the function's specifications

2. The determination of output based on the function's specifications
3. The execution of the test case
4. The comparison of actual and expected outputs

PERFORMANCE TESTING

In software engineering, **performance testing** is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the implementation, design and architecture of a system.

TESTING TYPES

LOAD TESTING

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards bottlenecks in the application software.

STRESS TESTING

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

SOAK TESTING

Soak testing, also known as endurance testing, is usually done to determine if the system can sustain the continuous expected load. During soak tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended, significant period of time. The goal is to discover how the system behaves under sustained use.

SPIKE TESTING

Spike testing is done by suddenly increasing the number of or load generated by, users by a very large amount and observing the behaviour of the system. The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load.

CONFIGURATION TESTING

Rather than testing for performance from the perspective of load, tests are created to determine the effects of configuration changes to the system's components on the system's performance and behaviour. A common example would be experimenting with different methods of load-balancing.

ISOLATION TESTING

Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Often used to isolate and confirm the fault domain.

INTEGRATION TESTING

Integration testing (sometimes called **integration and testing**, abbreviated **I&T**) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

PURPOSE

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.

Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up. Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.

BIG BANG

In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

A type of Big Bang Integration testing is called **Usage Model testing**. Usage Model Testing can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use.

Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

For integration testing, Usage Model testing can be more efficient and provides better test coverage than traditional focused functional integration testing. To be more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

TOP-DOWN AND BOTTOM-UP

Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

Sandwich Testing is an approach to combine top down testing with bottom up testing.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link.

VERIFICATION AND VALIDATION

Verification and Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fully fills its intended purpose. These are critical components of a quality management system such as ISO 9000. The words "verification" and "validation" are sometimes preceded with "Independent" (or IV&V), indicating that the verification and validation is to be performed by a disinterested third party.

It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?" In practice, the usage of these terms varies. Sometimes they are even used interchangeably.

The PMBOK guide, an IEEE standard, defines them as follows in its 4th edition

- **"Validation.** The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with *verification*."
- **"Verification.** The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with *validation*."
- Verification is intended to check that a product, service, or system (or portion thereof, or set thereof) meets a set of initial design specifications. In the development phase, verification procedures involve performing special tests to model or simulate a portion, or the entirety, of a product, service or system, then performing a review or analysis of the modelling results. In the post-development phase, verification procedures involve regularly repeating tests devised specifically to ensure that the product, service, or system continues to meet the initial design requirements, specifications, and regulations as time progresses. It is a process that is used to evaluate whether a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. Verification can be in development, scale-up, or production. This is often an internal process.
- Validation is intended to check that development and verification procedures for a product, service, or system (or portion thereof, or set thereof) result in a product, service, or system (or portion thereof, or set thereof) that meets initial

requirements. For a new development flow or verification flow, validation procedures may involve modelling either flow and using simulations to predict faults or gaps that might lead to invalid or incomplete verification or development of a product, service, or system (or portion thereof, or set thereof). A set of validation requirements, specifications, and regulations may then be used as a basis for qualifying a development flow or verification flow for a product, service, or system (or portion thereof, or set thereof). Additional validation procedures also include those that are designed specifically to ensure that modifications made to an existing qualified development flow or verification flow will have the effect of producing a product, service, or system (or portion thereof, or set thereof) that meets the initial design requirements, specifications, and regulations; these validations help to keep the flow qualified. It is a process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders. This is often an external process.

- It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?". "Building the right thing" refers back to the user's needs, while "building it right" checks that the specifications are correctly implemented by the system. In some contexts, it is required to have written requirements for both as well as formal procedures or protocols for determining compliance.
- It is entirely possible that a product passes when verified but fails when validated. This can happen when, say, a product is built as per the specifications but the specifications themselves fail to address the user's needs.

Activities

VERIFICATION of machinery and equipment usually consists of design qualification (DQ), installation qualification (IQ), operational qualification (OQ), and performance qualification (PQ). DQ is usually a vendor's job. However, DQ can also be performed by the user, by confirming through review and testing that the equipment meets the written acquisition specification. If the relevant document or

manuals of machinery/equipment are provided by vendors, the later 3Q needs to be thoroughly performed by the users who work in an industrial regulatory environment. Otherwise, the process of IQ, OQ and PQ is the task of validation. The typical example of such a case could be the loss or absence of vendor's documentation for legacy equipment or do-it-yourself (DIY) assemblies (e.g., cars, computers etc.) and, therefore, users should endeavour to acquire DQ document beforehand. Each template of DQ, IQ, OQ and PQ usually can be found on the internet respectively, whereas the DIY qualifications of machinery/equipment can be assisted either by the vendor's training course materials and tutorials, or by the published guidance books, such as *step-by-step* series if the acquisition of machinery/equipment is not bundled with on- site qualification services. This kind of the DIY approach is also applicable to the qualifications of software, computer operating systems and a manufacturing process. The most important and critical task as the last step of the activity is to generating and archiving machinery/equipment qualification reports for auditing purposes, if regulatory compliances are mandatory.

Qualification of machinery/equipment is venue dependent, in particular items that are shock sensitive and require balancing or calibration, and re-qualification needs to be conducted once the objects are relocated. The full scales of some equipment qualifications are even time dependent as consumables are used up (i.e. filters) or springs stretch out, requiring recalibration, and hence re-certification is necessary when a specified due time lapse Re-qualification of machinery/equipment should also be conducted when replacement of parts, or coupling with another device, or installing a new application software and restructuring of the computer which affects especially the pre-settings, such as on BIOS, registry, disk drive partition table, dynamically-linked (shared) libraries, or an ini file etc., have been necessary. In such a situation, the specifications of the parts/devices/software and restructuring proposals should be appended to the qualification document whether the parts/devices/software are genuine or not.

Torres and Hyman have discussed the suitability of non-genuine parts for clinical use and provided guidelines for equipment users to select appropriate substitutes which are capable to avoid adverse effects. In the case when genuine parts/devices/software are demanded by some of regulatory requirements, then re-

qualification does not need to be conducted on the non-genuine assemblies. Instead, the asset has to be recycled for non-regulatory purposes.

When machinery/equipment qualification is conducted by a standard endorsed third party such as by an ISO standard accredited company for a particular division, the process is called certification. Currently, the coverage of ISO/IEC 15408 certification by an ISO/IEC 27001 accredited organization is limited; the scheme requires a fair amount of efforts to get popularized.

SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification

Types of tests to include in system testing

The following examples are different types of testing that should be considered during System testing:

- Graphical user interface testing
- Usability testing

- Software performance testing
- Compatibility testing
- Exception handling
- Load testing
- Volume testing
- Stress testing
- Security testing
- Scalability testing
- Sanity testing
- Smoke testing
- Exploratory testing
- Ad hoc testing
- Regression testing
- Installation testing
- Maintenance testing Recovery testing and failover testing.
- Accessibility testing, including compliance with:
 - Americans with Disabilities Act of 1990
 - Section 508 Amendment to the Rehabilitation Act of 1973
 - Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

STRUCTURE TESTING:

It is concerned with exercising the internal logic of a program and traversing particular execution paths.

OUTPUT TESTING:

- Output of test cases compared with the expected results created during design of test cases.
- Asking the user about the format required by them tests the output generated or displayed by the system under consideration.

- Here, the output format is considered into two was, one is on screen and another one is printed format.
- The output on the screen is found to be correct as the format was designed in the system design phase according to user needs.
- The output comes out as the specified requirements as the user's hard copy.

USER ACCEPTANCE TESTING:

- Final Stage, before handing over to the customer which is usually carried out by the customer where the test cases are executed with actual data.
- The system under consideration is tested for user acceptance and constantly keeping touch with the prospective system user at the time of developing and making changes whenever required.
- It involves planning and execution of various types of test in order to demonstrate that the implemented software system satisfies the requirements stated in the requirement document.

TWO SET OF ACCEPTANCE TEST TO BE RUN:

1. Those developed by quality assurance group.
2. Those developed by customer.

CHAPTER 9

9. CONCLUSION

CONCLUSION

The software puzzle may be built upon a data puzzle, it can be integrated with any existing server-side data puzzle scheme, and easily deployed as the present client puzzle schemes do. CAPTHCHA is widely research field act as internet rectifier to secure web applications by discern human from bots. CAPTCHA presented which will improve resistance of math calculus CAPTCHA. By use, Boolean operations and expressions instead of trigonometric and differential function which will help in reduce the complexity of CAPTCHA and help to achieve better usability and security

as compared to math calculus CAPTCHA. Boolean CAPTCHA can be easily use by educated user. No need of technical skill, by using intellectual mind to solve this CAPTCHA and help to reduce time complexity.

CHAPTER 10

10. FUTURE WORK

FUTURE WORK:

Authors propose using machine learning classifiers to attacks captchas. In the same authors study how efficient statistical classifier are at recognizing captcha letters. In the authors study how good humans are at solving well-known captchas using Mechanical Detecting and removing lines is a well -studied field in computer vision since the '70s. Two well-known and efficient algorithms that can be used against captchas with lines are the Canny detection and the Hough Transform Removing noise using a Markov Random Field (Gibbs) was introduced in Many image descriptors have been proposed over the last decades: one of the first and most used descriptors is the the Harris Corner detector introduced . However, recently it has been replaced by more complex descriptors that are insensitive to scale and rotation (to a certain extent).

CHAPTER 11

11. REFERENCES

REFERENCES

- ✓ Manal Adham¹, Amir Azodi^{1;3}, Yvo Desmedt² “How to Attack Two-Factor Authentication Internet Banking”.
- ✓ Fadi Aloul, Syed Zahid Department of Computer Science “Two Factor Authentication Using Mobile Phones”
- ✓ Fadi Aloul, Syed Zahid Department of Computer Science “Users are not the enemy”

- ✓ Robert Biddle, Sonia Chiasson, P.C. van Oorschot School of Computer Science Carleton University, "Graphical Password: Learning from the First Twelve Years"
- ✓ Joseph Bonneau University of Cambridge, "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes"

CHAPTER 12

12. APPENDIX

APPENDIX:

12.1 SOURCE CODE:

```
<project xmlns:webproject1="http://www.netbeans.org/ns/web-project/1"
xmlns:webproject2="http://www.netbeans.org/ns/web-project/2"
xmlns:webproject3="http://www.netbeans.org/ns/web-project/3" basedir=".."
default="default" name="SoftwarePuzzle-impl">
  <import file="ant-deploy.xml"/>
  <fail message="Please build using Ant 1.7.1 or higher.">
    <condition>
      <not>
        <antversion atleast="1.7.1"/>
      </not>
    </condition>
  </fail>
  <target depends="dist,javadoc" description="Build whole project."
name="default"/>
  <!--
      INITIALIZATION SECTION
  -->
  <target name="-pre-init">
    <!-- Empty placeholder for easier customization. -->
    <!-- You can override this target in the ../build.xml file. -->
```

```

</target>
<target depends="-pre-init" name="-init-private">
  <property file="nbproject/private/private.properties"/>
</target>
<target depends="-pre-init,-init-private" name="-init-user">
  <property file="{user.properties.file}"/>
  <!-- The two properties below are usually overridden -->
  <!-- by the active platform. Just a fallback. -->
  <property name="default.javac.source" value="1.4"/>
  <property name="default.javac.target" value="1.4"/>
</target>
<target depends="-pre-init,-init-private,-init-user" name="-init-project">
  <property file="nbproject/project.properties"/>
</target>
<target depends="-pre-init,-init-private,-init-user,-init-project,-init-macrodef-
property" if="dist.ear.dir" name="-do-ear-init"/>
<target depends="-pre-init,-init-private,-init-user,-init-project,-init-macrodef-
property" name="-do-init">
  <condition property="have.tests">
    <or>
      <available file="{test.src.dir}"/>
    </or>
  </condition>
  <condition property="have.sources">
    <or>
      <available file="{src.dir}"/>
    </or>
  </condition>
  <condition property="netbeans.home+have.tests">
    <and>
      <isset property="netbeans.home"/>
      <isset property="have.tests"/>
    </and>
  </condition>

```

```

<condition property="no.javadoc.preview">
  <isfalse value="${javadoc.preview}"/>
</condition>
<property name="javac.compilerargs" value=""/>
<condition property="no.deps">
  <and>
    <istrue value="${no.dependencies}"/>
  </and>
</condition>
<condition property="no.dist.ear.dir">
  <not>
    <isset property="dist.ear.dir"/>
  </not>
</condition>
<property name="build.web.excludes" value="${build.classes.excludes}"/>
<condition property="do.compile.jsps">
  <istrue value="${compile.jsps}"/>
</condition>
<condition property="do.debug.server">
  <or>
    <not>
      <isset property="debug.server"/>
    </not>
    <istrue value="${debug.server}"/>
  <and>
    <not>
      <istrue value="${debug.server}"/>
    </not>
    <not>
      <istrue value="${debug.client}"/>
    </not>
  </and>
</or>
</condition>

```

```

<condition property="do.debug.client">
  <istrue value="\${debug.client}"/>
</condition>
<condition property="do.display.browser">
  <istrue value="\${display.browser}"/>
</condition>
<condition property="do.display.browser.debug.old">
  <and>
    <isset property="do.display.browser"/>
    <not>
      <isset property="do.debug.client"/>
    </not>
    <not>
      <isset property="browser.context"/>
    </not>
  </and>
</condition>
<condition property="do.display.browser.debug">
  <and>
    <isset property="do.display.browser"/>
    <not>
      <isset property="do.debug.client"/>
    </not>
    <isset property="browser.context"/>
  </and>
</condition>
<available file="\${conf.dir}/MANIFEST.MF" property="has.custom.manifest"/>
<available file="\${persistence.xml.dir}/persistence.xml"
property="has.persistence.xml"/>
<condition property="do.war.package.with.custom.manifest">
  <isset property="has.custom.manifest"/>
</condition>
<condition property="do.war.package.without.custom.manifest">
  <not>

```

```

        <isset property="has.custom.manifest"/>
    </not>
</condition>
<condition property="do.tmp.war.package.with.custom.manifest">
    <and>
        <isset property="has.custom.manifest"/>
        <or>
            <isfalse value="\${directory.deployment.supported}"/>
            <isset property="dist.ear.dir"/>
        </or>
    </and>
</condition>
<condition property="do.tmp.war.package.without.custom.manifest">
    <and>
        <not>
            <isset property="has.custom.manifest"/>
        </not>
        <or>
            <isfalse value="\${directory.deployment.supported}"/>
            <isset property="dist.ear.dir"/>
        </or>
    </and>
</condition>
<condition property="do.tmp.war.package">
    <or>
        <isfalse value="\${directory.deployment.supported}"/>
        <isset property="dist.ear.dir"/>
    </or>
</condition>
<property name="build.meta.inf.dir" value="\${build.web.dir}/META-INF"/>
<condition else="" property="application.args.param"
value="\${application.args}">
    <isset property="application.args"/>

```

12.2 SCREEN SHOTS

The screenshot shows a web browser window with the title "SoftWare Puzzle". The address bar displays "localhost:8080/SoftwarePuzzle/RegisterLoginForm.jsp". The page content is a "User Registration Form" on a teal background. The form fields are as follows:

- Username: maha
- Password: ****
- DATE OF BIRTH: 5 Mar 2002
- GENDER: FEMALE
- Age: 20
- Email: mahabowya2001@gmail.co
- Phone: 9090909090
- City: chennai
- State: chennai
- Country: tamil nadu
- Country: Indian
- Submit button

The Windows taskbar at the bottom shows the search bar, task view button, and various application icons. The system tray on the right indicates the time as 15:10 on 10-03-2022.

