

Online Electronic Payment System Using PREDICT II Protocol

Nishanth Menon

8th October 2001

Contents

1	Introduction	1
1.1	Background	1
1.2	The Electronic Payment Systems	1
1.3	Through the Looking Glass	1
2	System Analysis	3
2.1	Introduction	3
2.2	The Problem Statement	3
2.2.1	Inputs and Outputs.	3
2.3	Present Systems for EPS	4
2.3.1	Adapting Existing Methods	4
2.3.2	Digital Currencies	4
2.3.3	Characteristics	4
2.3.4	Limitations	4
2.3.5	Advantages	5
2.4	Proposed System	5
2.4.1	Introduction	5
2.4.2	Assumptions for the PREDICT II	5
2.4.3	Description	5
2.5	Functional Requirements for the Project	6
3	Design	7
3.1	Introduction	7
3.2	The Conceptual Design	7
3.2.1	System Use Case	7
3.2.2	Responsibilities of Various Entities	7
3.2.3	Description	9
3.3	The Messages	9
3.3.1	Introduction	9
3.3.2	Messages	10
3.3.3	The Sequence Diagram	12
3.4	The Design of the Client Software	12
3.4.1	The Class Design	12
3.5	The Design of the Zone Server Software	14
3.5.1	The Class Description	15
3.6	The Design of the Master Server Software	15
3.6.1	The Class Diagram of the Master Server	15
3.7	The DataBase Design	18
3.7.1	The Design of the Zone Server DataBase	18
3.7.2	The Design of the Master Server Database	19

3.8	The Package Layout	19
3.8.1	Main Packages	19
3.8.2	The Helper Packages	21
4	Implementation	22
4.1	Introduction	22
4.2	The Working of the Client	22
4.2.1	The Working of the Client as a Citizen	22
4.2.2	The Working of the Client as a Merchant	22
4.3	The Working of the Zone Server	22
4.4	The Working of the Master Server	26
4.4.1	Zone Server Validator	26
4.4.2	Zone Server Depositor	26
4.4.3	Master Server Remote Validator	26
4.5	Highlights of the Implementation	28
4.5.1	Asynchronous Network Transmission	28
4.5.2	Multi Threading	28
4.5.3	Data Base Concurrency	28
4.5.4	Time Analysis	29
5	Testing	30
5.1	Introduction	30
5.2	Test Plan	30
5.3	Unit Testing	30
5.4	Integration Testing	30
5.5	Validation Testing	30
5.6	Output Testing	31
5.7	Conclusion	31
6	Conclusion	32
6.1	Introduction	32
6.2	Limitations of the system	32
6.3	Advantages of the system	32
6.4	Future Work	33
	Bibliography	34

Abstract

Electronic Payment System is the new buzzword in the market, with every Tom, Dick and Harry offering services related to this largely uncharted arena. Unfortunately, this specific area consists of innumerable issues making it even more murky. Hence, most of the startup firms mushroom and fail in matter of years. My analysis of this and similar aspects of the issue has resulted in my own protocol called the PProtocol Enhancement for DIgital Cash Transaction (PREDICT). The second version of this protocol has been used for this project. This report speaks about this project in detail while explaining only the necessary aspects of PREDICT. An unorthodox view has to be taken to understand my line of thought. This is because, I consider that a transaction happens only when a transfer of wealth takes place; while, the tools and techniques used for this transfer are of secondary importance. In a nutshell, each user has an account with the zone server. They have to contact the zoneserver for any transaction. They may withdraw money or deposit money with the zone server, at the same time they may recieve money or give money to a remote user. Their personal transaction information is recieved from the web server maintained by the zone server. For the facilitation of inter-zone transmission of money, there are entities called Master servers that help maintain the integrity of the transactions. In short, this project is all about how to send money from one person to another.

1 Introduction

He who tampers with the currency robs labour of its bread
- Daniel Webster

1.1 Background

Monetary transactions have evolved from their ancient forms such as barter, to today's payment systems. However, the sheer volume of transactions in today's commerce is creating an enormous strain on the already over-exploited resources of the world. The human race thus has two options - either look for more resources for maintaining the present-day style of transactions, or evolve today's payment systems to come to terms with tomorrow's technology.

1.2 The Electronic Payment Systems

The Internet is a rapidly evolving information infrastructure ("Information Highway") which provides global connectivity, easy reachability and interactive communications at moderate cost to the consumer. The dominating application is the World Wide Web (WWW), with its potential of 3 million connected computer systems and an order of magnitude more actual users. Currently, the WWW is primarily used to provide easy access to free-of-charge information (typically research or marketing information). But this is expected to change dramatically in the near future. The WWW is expected to provide a basis for electronic commerce and trade. A similar development can be expected for broadband networks and "Information Highways".

The development of various Electronic Payment Systems (EPS) has made a valuable contribution in this direction. In short, EPS is the logical evolution of the payment systems that exist today so that any form of commerce exists in the technology of tomorrow.

The research into EPS is still in its infancy. The popularity of these payment systems can be seen by the fact that the first commercial Digital cash venture, DigiCash, started up in 1994 and filed for bankruptcy in 1998, after a mere four years of existence.

Most of the focus has been on credit card-based transactions, with the interest into anonymous payment systems having not been more than an academic interest in recent years (especially after the US economy began its recession). However, the concept of the credit card has not had much success in replacing the entire existing payment system based on paper money, due to the lack of adequate amount of security and scalability to handle the sheer volume of today's commerce.

1.3 Through the Looking Glass

This report deals with how this project was done. This includes the analysis, design and implementation details of the project.

This system is basically envisaged as an interaction between the monetary system and the user.

The user has a software called the client (in my terminology the *Kuchelan*¹) which interacts with the zone server software (I call it the *Kuberan*) who requests information from the master server (again in the same spirit called the *Brahma*).

The following is a brief overview on this report.

Chapter 2 (p 3) deals with the analysis of the system and discusses how the present architecture has been reached.

Chapter 3 (p 7) deals with the design of the system from the analysis we have made. This is where my focus has been in this project, as a perfect design results in a product that requires very little modification and contains minimal errors. Starting from the very basic framework of the system, the chapter leads to a moderately detailed description of the system.

Chapter 4 (p 22) deals with the implementation and the issues faced in this. The issues, remarkably, from my previous experience, have been minimal. I have not stated the HTTPS setup here as a moderate experience in Linux should help anyone setup their own certification for the purpose of testing. All the components for the transaction are embedded in a Java application, while only the queries about a specific user have been allocated for the web end.

Chapter 5 (p 30) deals with how the system was tested. I have indeed spent some effort in making sure that there are as few bugs in the software as possible.

Chapter 6 (p 32) deals with my conclusion and my personal views on the subject as to what has to be done in the project. In fact, this could create a report on its own and hence, I have restricted myself to only the major items.

¹The terms kuchelan, kuberan and brahma come from the Indian mythology. Kuchelan was the destitute who having no money went to Lord Krishna for a bowl of rice. Kuberan is supposed to have been the king of Lanka before Ravana took over control. He is supposed to also have been so rich as to be a banker of the Lords. The term Brahma, comes from Lord Brahma, the lord of creation of the universe. The significance is that kuberan supplies the cash token, while brahma controls them and kuchelan is that part of the system that requests for money.

2 System Analysis

If a man take no thought about what is distant, he will find sorrow near at hand.
-Confucius

2.1 Introduction

This section analyses the system as such. To analyze the monetary transaction in its fullest including the fallouts of governmental policies is beyond the scope of this report. Hence, only the technical aspect of the problem is analyzed here. This section analyses current systems in existence for the EPS. We then speak of their failures and strengths.

2.2 The Problem Statement

The objective of this project is to create a scalable model of an online Electronic Payment System, allowing authorized users to access and utilize their wealth from anywhere on the Internet using the PProtocol Enhancement of DIgital Cash Transaction- II protocol.

2.2.1 Inputs and Outputs.

2.2.1.1 System Inputs

1. User Information such as UserName, Pass Phrase etc.
2. The amount of Cash to be transacted.
3. The IP of the Zone Server.
4. The IP and optionally the port of the Server socket of the remote client software.
5. The Key Generation information (including the number of bits of the RSA Key, Hashing algorithm for the password storage, Hashing algorithm for encrypting the RSA Key, Encryption algorithm for storing the RSA Key, other configuration information).

2.2.1.2 System Outputs

1. The result of the transaction – Whether completed or failed etc.
2. Passbook format transaction information.
3. Complete transaction information.
4. Personal information of authorized client users.

2.3 Present Systems for EPS

A modern customer adopts different methods, like cash, credit cards, personal checks etc., to pay for goods and services. Internet-based electronic commerce methods also focus on the secure transmission of credit card information, electronic checking and digital currencies.

2.3.1 Adapting Existing Methods

Credit cards are the easiest to adapt to on-line transactions, because people are accustomed to using them remotely. Credit card transactions simply require that the customer provide a valid credit card number and the expiry date while placing an order through e-mail etc. To secure these information from eavesdroppers, encryption is applied.

People associate cash with the physical exchange of currency. Though it makes possible to spend money anonymously, it is difficult to be used over an open network. For this purpose, the cash is digitized. While digitizing cash, the actual currency is replaced by 'digital coins' represented as chunks of data. Digital currencies are the electronic counterparts of traditional currency.

2.3.2 Digital Currencies

These are intended to carry value in a protected digital form over the Internet. Anyone can participate by opening an account with a financial institution offering digital currency services. Client software is used to withdraw money from the account, to check the balances and to maintain a digital wallet that holds the value on the participant's computer. Among the present payment systems are NetCash, DigiCash (later renamed as ECash), AMDIGI (UK), Avant(Finland), Beenz(UK and US), bibit (Denmark), The Common Electronic Purse Specifications (CEPS), CheckFree, Chipper, klikpay, PMTP and many similar systems.

2.3.3 Characteristics

1. Transferable without physical interaction.
2. They can be used to maintain the anonymity of the customer in an online transaction.
3. They can support an actual transfer of value by themselves unlike in the case of credit cards.
4. They provide support for micro payments.

2.3.4 Limitations

1. Security measures against spoof banks and other possible 'man in the middle' unwanted intermediaries have not been completely successful.
2. Requires a database to be kept that is un-scalable as the number of users increase.
3. The necessity of having security inbuilt into the system has made most systems inherently dependent on the encryption technique used. At times, as in the case of 'Beenz' payment system used in Finland, the discovery of some failure of the same encryption technique results in the abrupt obsolescence of the system.
4. Some of the payment systems are not anonymous and the financial institutions are aware of the transactions taking place.
5. Cash has to e-stored in a cyber wallet and hence if your computer crashes or your hard disk gets stolen, well, that is it.

6. A governmental nightmare is the possibility of untold amount of wealth hidden away yet useable, and with absolutely no traceability.
7. Many EPS suffer from the probability of double payment, in which one user sends the cash token to more than one user.

2.3.5 Advantages

1. The freedom of cash transactions being online and instantaneous in nature.
2. Possibility of the cash transactions being done across international borders possibly with a minimal amount of fuss.
3. More secure than most payment systems that may retain your credit card number and inherent anonymity is present.

2.4 Proposed System

2.4.1 Introduction

The above mentioned problems have existed for some time now. Unfortunately, no single system existed that could address this problem. PProtocol Enhancement of DIgital Cash Transaction- II (PREDICT II) tries to address these issues. The PREDICT I version had inherent security failures that compromised the security and minimal anonymity offered by the system. It has been revised and the revised form has been used in this project.

2.4.2 Assumptions for the PREDICT II

1. Network authentication and secure communications used in all parts of the Internet.
2. Possibility of extremely secure and low-failure custom network that can be used for the internal transmissions.
3. Possibility of high-security servers that are secure enough to carry highly sensitive information such as cash remaining per citizen under a responsible controlling authority such as the government.
4. Existence of a high-bandwidth network. This is a tradeoff we have to make to achieve anonymity, scalability and security.
5. High performance and extremely reliable servers with redundant lines of communication can be set up.

2.4.3 Description

This system envisages the existence of *zone servers* that provide services to the users connecting to it from the internet. All clients that come under a single zone server are said to be under a *zone server zone*. For the purpose of logistics, a set of these zone servers come under a *master server*. This set of zone servers are said to be under a *master server zone*.

Each Master Server zone has a *Master Server ID*. The entire set of zone servers and clients under them are known by this ID.

A cash token is either withdrawn or deposited to a zone server by a user. If the user withdraws cash, then he/she is known as a *citizen* for the period of the transaction. If the user deposits cash then

he/she is known as a *merchant*, again only for the period of the transaction. So, the user may either be a citizen or a merchant depending on the role he/she plays in the transaction.

A cash token once deposited results in an acknowledgement being sent to the merchant who forwards it to the citizen and then it gets forwarded on to the citizen's zone server. Then both the citizen and merchant get a transaction completion message. I further discuss this while explaining the design of the system.

2.5 Functional Requirements for the Project

1. *Creating an account* : The System should be able to create an account with the Zone Server for each authorized user.
2. *Deleting an account* : The System should be able to delete an account with the Zone Server of an authorized user.
3. *Requesting Passbook Information* : The System should provide information in Passbook format for an authorized client.
4. *Requesting Transaction History* : The System should provide an authorized client, on request, information about past transactions.
5. *Change of Personal Information* : The System should provide ability for a valid client to change his personal information.
6. *Deciding on the amount of cash to be transacted* : The client has the ability to decide on the amount of cash to be transacted in the system.
7. *Sending Cash token* : The client must be able to send a cash token to a remote client.
8. *Receipt of Cash token* : The client must be able to receive a cash token from a remote client.

3 Design

Give me matter, and I will construct a world out of it!

-Immanuel Kant

3.1 Introduction

The design of the system was done in the best traditions of the Object Oriented philosophy. The main tool used for the design has been Rational Rose 2000. Further, the UML was used as the basic language for the design documents.

The objective was to develop the class diagrams and generate the code stubs which were to be later fleshed out in the implementation phase.

In this chapter, explanation is done only where deemed necessary since the diagrams themselves are self explanatory.

3.2 The Conceptual Design

The basic system may be identified as containing the elements as in Figure:3.2 (p 9). This represents the system in its entirety.

3.2.1 System Use Case

From the analysis, the system is represented as in the use case shown in 3.1 (p 8). This shows basically how the system layout has been. This specifically helps in the understanding the system.

The point to note is that a 'client' may be either a 'Citizen' – someone who is to send money, or a 'Merchant' – someone who is to receive money. Corporations or banks are clients with a special transaction allowance specified by the government. But all the entities are basically various forms of the client.

3.2.2 Responsibilities of Various Entities

The entities in the system have been identified as shown in Figure:3.2 (p 9). These entities have been decided on the basis of their responsibilities as defined below.

The responsibilities of the various entities are as follows.

Zone Server: The zone server has the responsibility to maintain the user information, to authenticate the clients connecting to it, to maintain the sequence of messages, to communicate with the master server when necessary and to do other similar operations as specified in PREDICT II.

Master Server: The Master server has the responsibility to maintain the identifier integrity when client zone servers and remote master servers contact it for information. At the same time, it has to maintain the integrity of the master server zones under it. Database integrity has to be maintained and when there is a probability that the transaction may be corrupted, it should abort the transaction

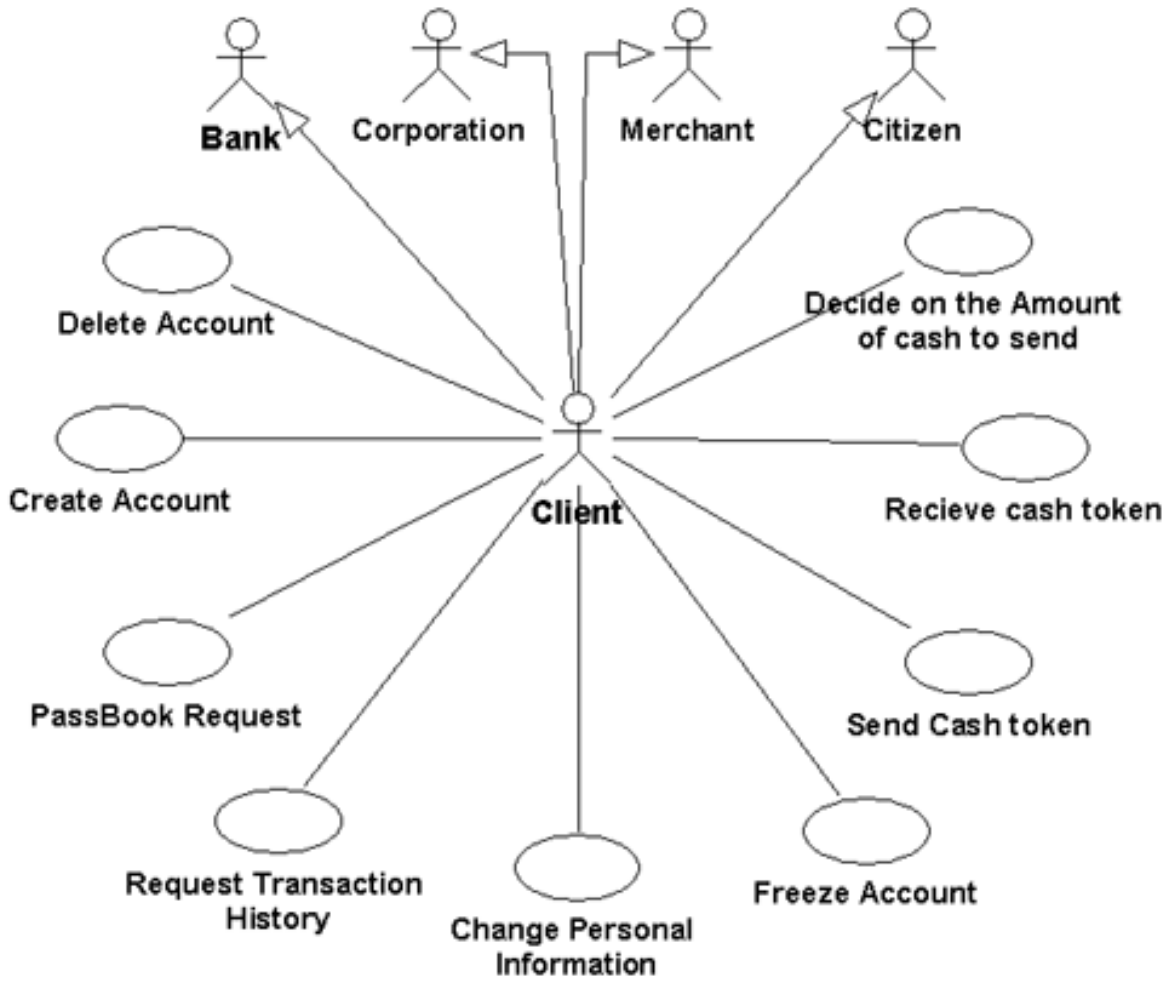


Figure 3.1: System Use Case Diagram

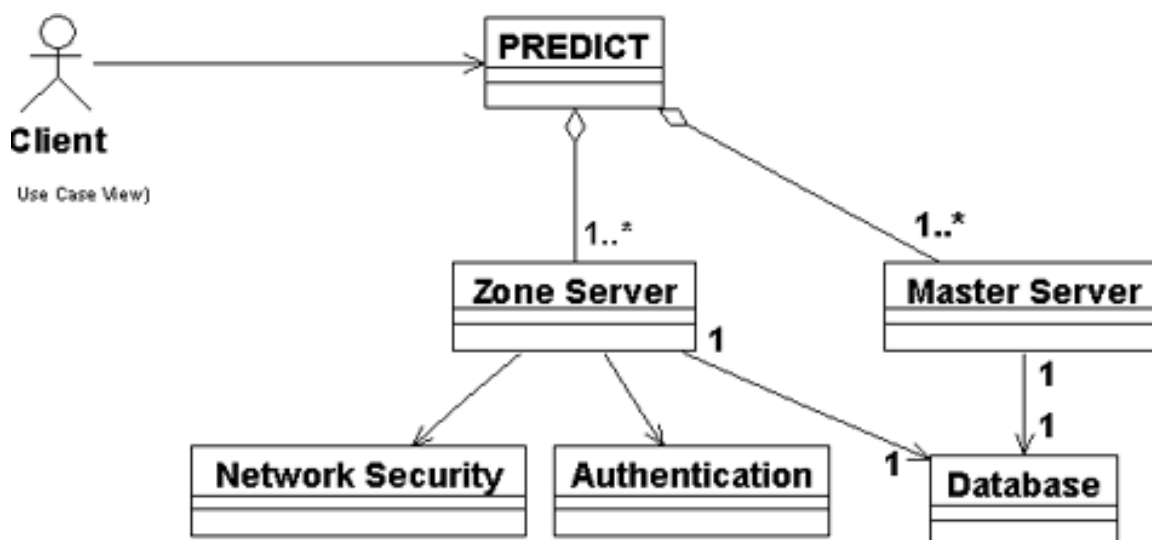


Figure 3.2: The Conceptual Class Diagram

Client: The client has the responsibility to follow the message patterns defined in PREDICT II. The client also has to generate a random number that is used in the identifier. Further, the client should analyse the network time for the transactions to be completed¹.

3.2.3 Description

The PREDICT system may instantiate the following classes. Each class represents a different role.

The *client* is the software that belongs to the user. This is used for interacting with the components of the PREDICT II system.

The *Zone Server* is that part of the system that has the responsibilities of the zone server as defined in PREDICT II. Each instance of the zone server may or may not include authentication and security classes but always includes the database class.

The *Master Server* is that part of the system that has the responsibilities of the master server as defined in PREDICT II.

Authentication is based on the use of a passphrase. This is done, not by actually sending the passphrase, but only its hash, to the server in question. This hash is stored in the database and compared at a later time, when the requirement for authentication of the client arises.

Network Security is an optional module in this design used only when the client in question works on a network that does not possess any security of its own. The decision to use or not to use this module rests solely with the zone server.

The *DataBase* class is a generalization of the various database related activities of the modules that call it. This has the responsibility of maintaining the integrity and the concurrency aspects of the system.

3.3 The Messages

3.3.1 Introduction

This section deals with the messaging techniques in the system. Messages are used for communication between the classes and the proper timing of these messages is crucial for the working of the

¹The time analysis done in this project is a complex issue and is dealt with in this report only in the conceptual form.

system.

3.3.2 Messages

3.3.2.1 Identifier

The Identifier is a String that uniquely identifies the cash token. This is useful for the database and integrity checks.

Structure

DENOMINATION	MASTERSERVER ID	IDENTIFIER
--------------	-----------------	------------

Default delimiter = ;;

3.3.2.2 Super Message Structure

The Super Message is the superset of all message bodies. This is used to identify the message and handle the parsing correctly. This super set may also contain timing information which may be used for the time analysis. I have not done it in this manner.

Structure

MESSAGE ABBREVIATION	MESSAGE CONTENTS
----------------------	------------------

Default delimiter = ||

3.3.2.3 Standard Messages

A Message is a special entity that describes some event in occurrence. Each message has a specific format specified by PREDICT II. This is shown in the table given below.

This Page is to be Printed by MSword. I am not able to get the Table format right here.

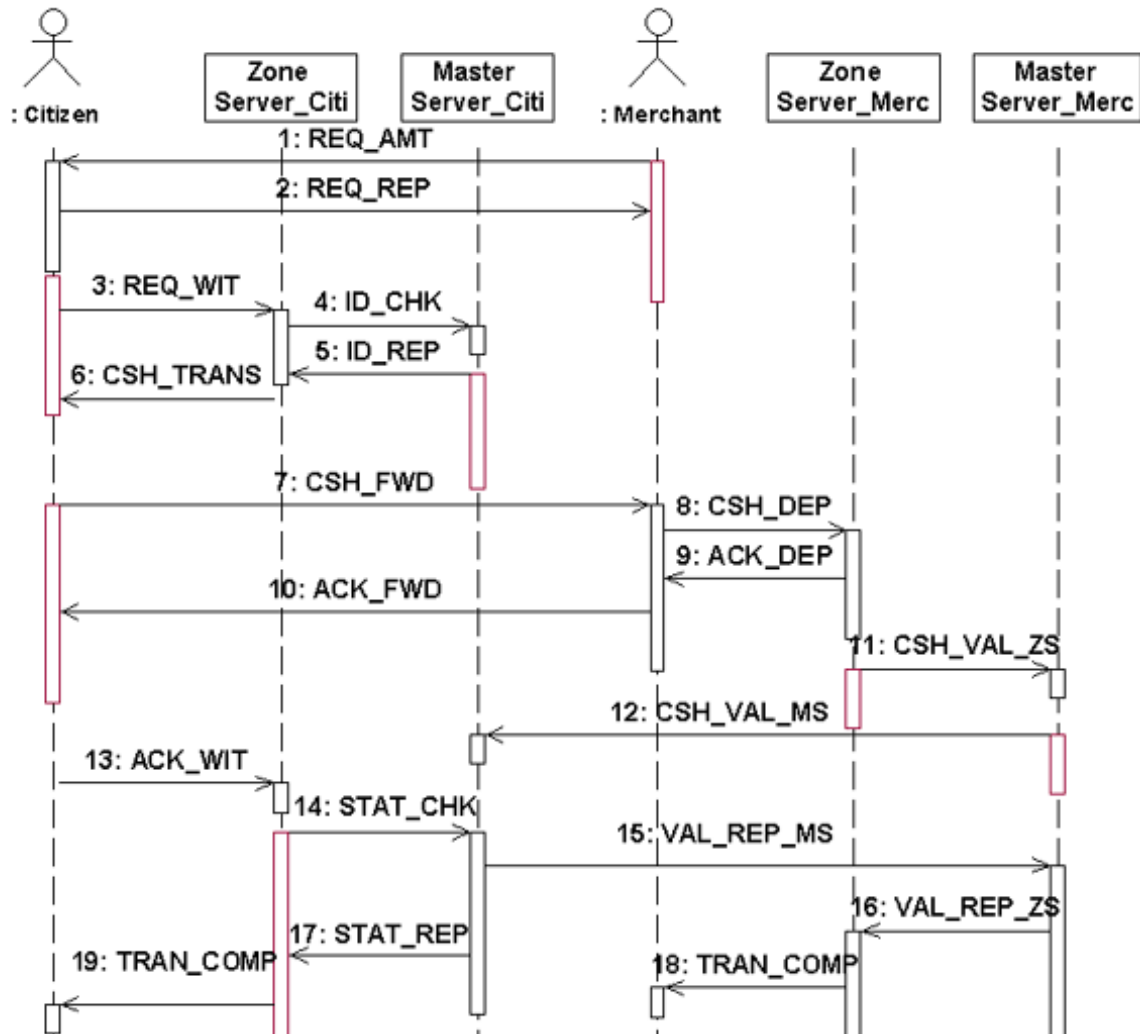


Figure 3.3: The Sequence Diagram of the Messages involved

3.3.3 The Sequence Diagram

We have seen the messages that are transmitted. Next, the diagram Figure 3.3 (p 12) describes the messages and the timings involved while transmitting the messages

3.4 The Design of the Client Software

The overall design of the client software is shown in Figure:3.4 (p 13). The requirements have been divided into the basic functional parts. Remote Clients are clients with whom the local client does the transaction.

3.4.1 The Class Design

The responsibilities of the functional units have been divided to the different classes as in Figure:3.5 (p 14).

The *kuchelan* class is the main class that initiates either the merchant or the citizen class based on the mode that the user selects.

The *Initializer* class loads up the essential information that is necessary for the functioning of the system. The class reads the zone server IP, generates the session key for the transaction, and handles

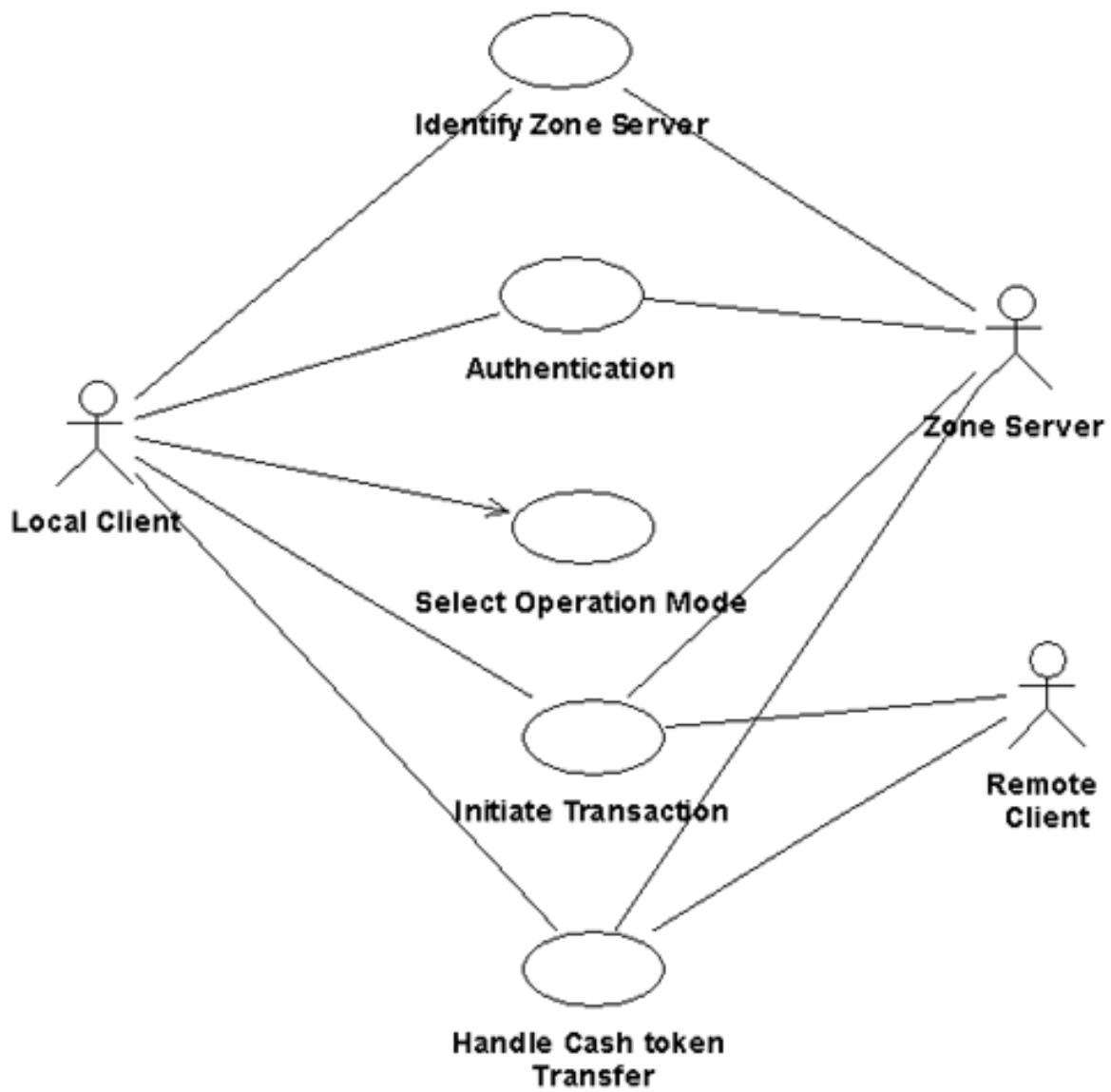


Figure 3.4: The Client Software Use Case

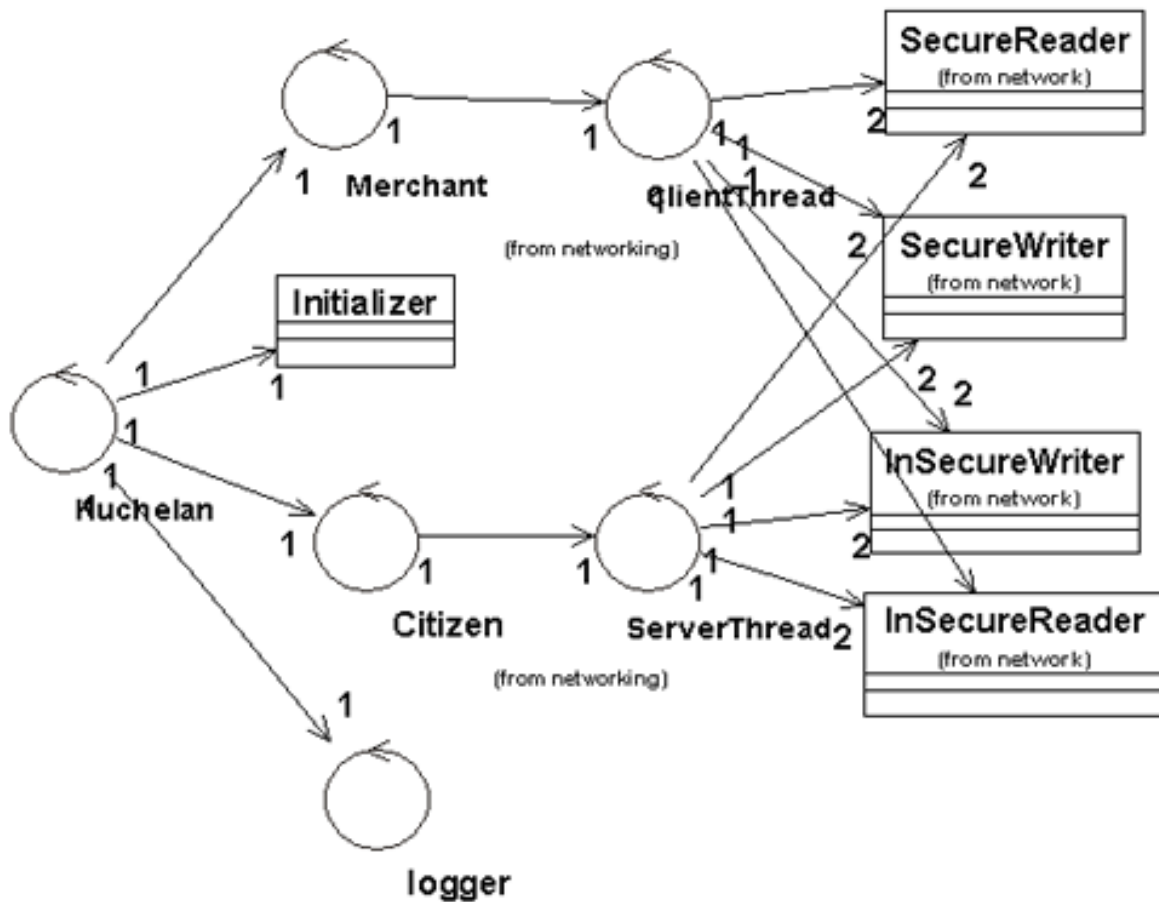


Figure 3.5: The Client Software Class Diagram

initial parameter requests by calling the appropriate classes in the GUI package.

The *Citizen* class does the initialization and waits for a client to contact it. If the user allows for the transaction to proceed, this invokes the *ServerThread* class and that thread runs independently.

The *ServerThread* class manages the messages that have to be handled by the citizen and generates the information required for the withdrawal of the cash token and successful completion of the transaction process. For this, it communicates with the zone server and proceeds as needed.

The *Merchant* class is invoked when the user decides to withdraw cash from a remote client. This initiates the *ClientThread* if and only if the connection is authorised.

The *ClientThread* handles the message sequence and other activities required of the Merchant. This communicates with the *ServerThread* of the remote client and its own zone server and proceeds as required.

3.5 The Design of the Zone Server Software

The Zone server software has the responsibility of satisfying user needs and providing for the services required by the zone server part of the PREDICT II protocol. The responsibilities have been divided into classes and displayed in Figure: 3.6 (p 15). This section discusses the various aspects of the zone server software and the responsibilities of the subclasses.

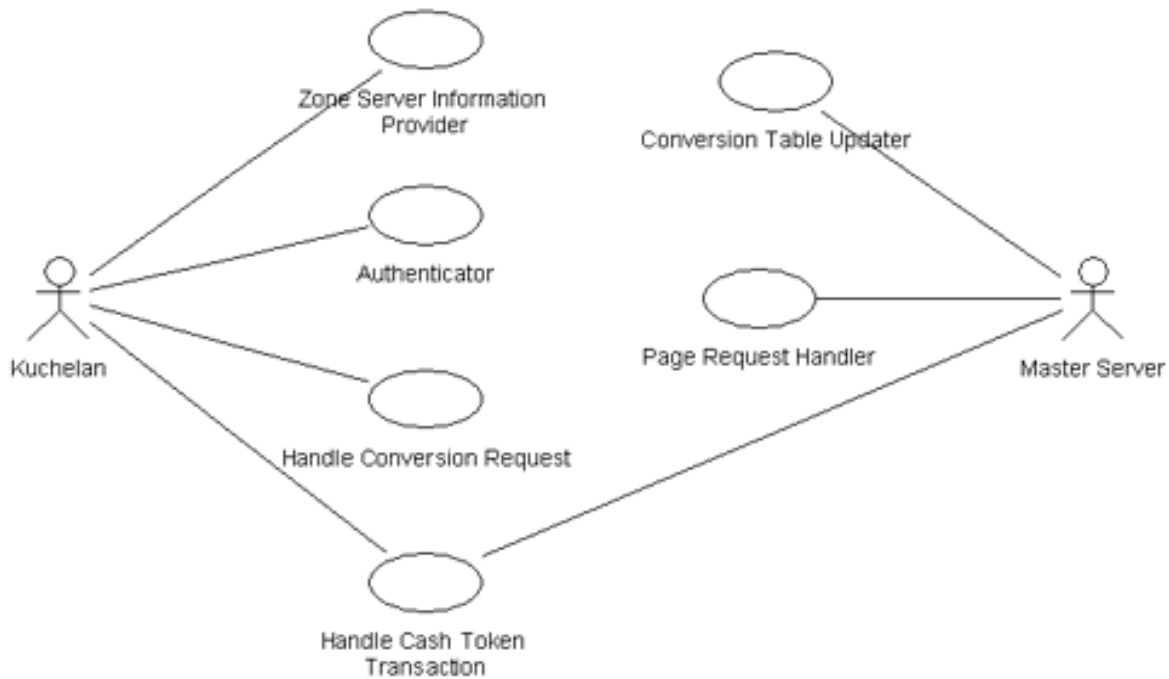


Figure 3.6: The Zone Server Use Case Diagram

3.5.1 The Class Description

This section discusses the main classes that create the components of the zone server software. The main classes and the relations between them is displayed in Figure: 3.7 (p 16).

The main class *kuchelan* instantiates the *HandleUserRequests* class when it starts up along with other miscellaneous threads required for maintenance. The *HandleUserRequest* class instantiates the *HandleAuthentication* class when any client connects to it. After the initial authentication, the zone server detects whether the client connected to it is a citizen or a merchant by the message sent to it. If the transaction starts with the message *REQ_WIT* then *HandleAuthentication* invokes the *HandleACitizen* class to handle the citizen, while if a *CSH_DEP* message arrives, it invokes an instance of the *HandleAMerchant* class to handle the Merchant.

The various helper classes used are those for the communication with the user and the master server when needed. The *InSecureReader* and *InSecureWriter* classes are used for communicating along the secure network to the Master Server for services required. The *SecureReader* and *SecureWriter* classes are used for communicating with the user on the insecure network. The *CheckDB* class is used as and when required to read needed information from the database.

3.6 The Design of the Master Server Software

The Master server is the entity that has to maintain the system's main interaction. This stores the identifiers that are requested to be deposited in it by the client zone server and at the same time provides for the verification and maintenance of this database.

This section deals with the description of how the system is made and how it actually works.

3.6.1 The Class Diagram of the Master Server

The *Brahma* class is the main class that on startup creates three service threads and other miscellaneous threads. The main class relations are shown in Figure:3.9 (p 18).

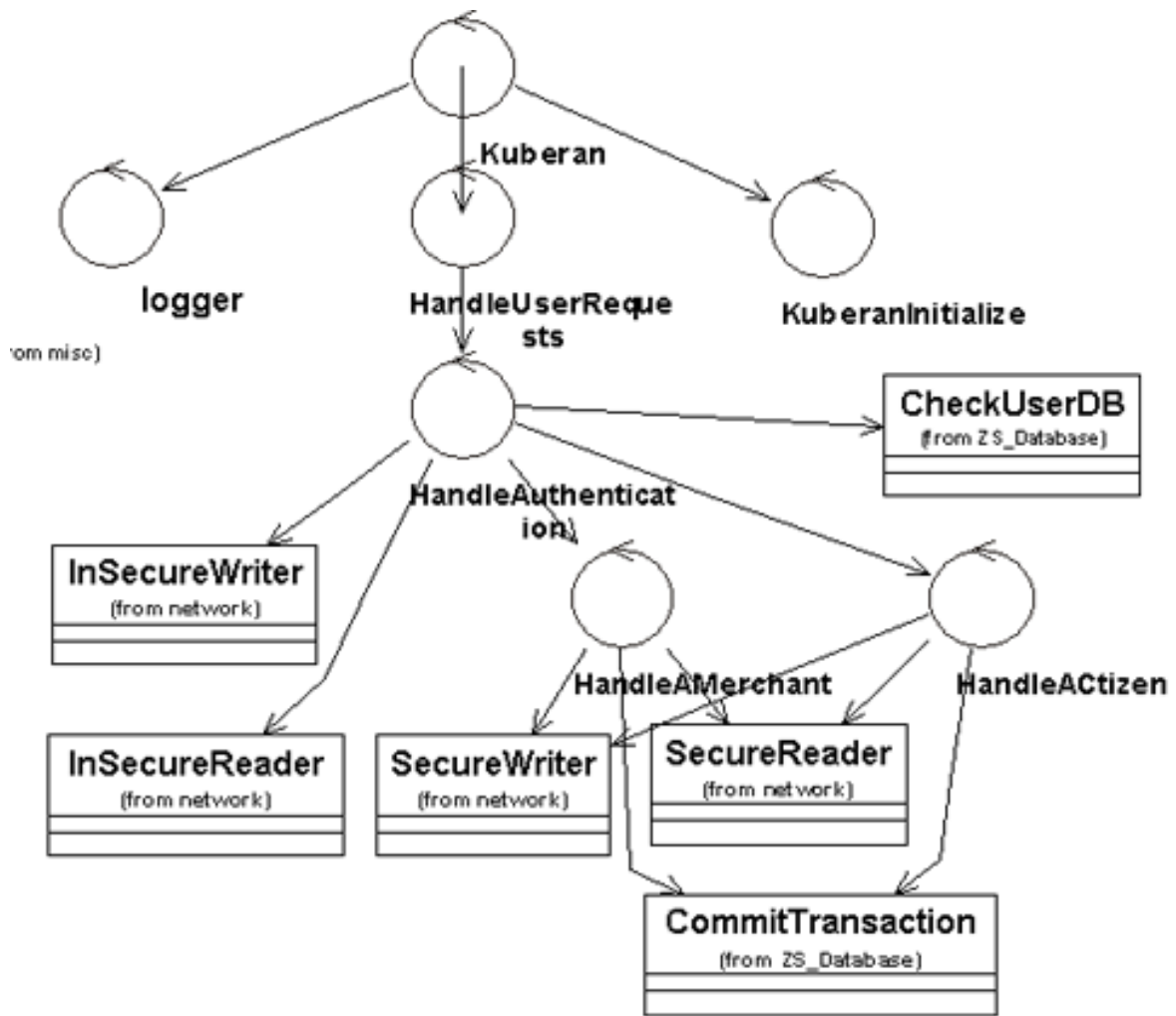


Figure 3.7: The Zone Server Software Class Diagram

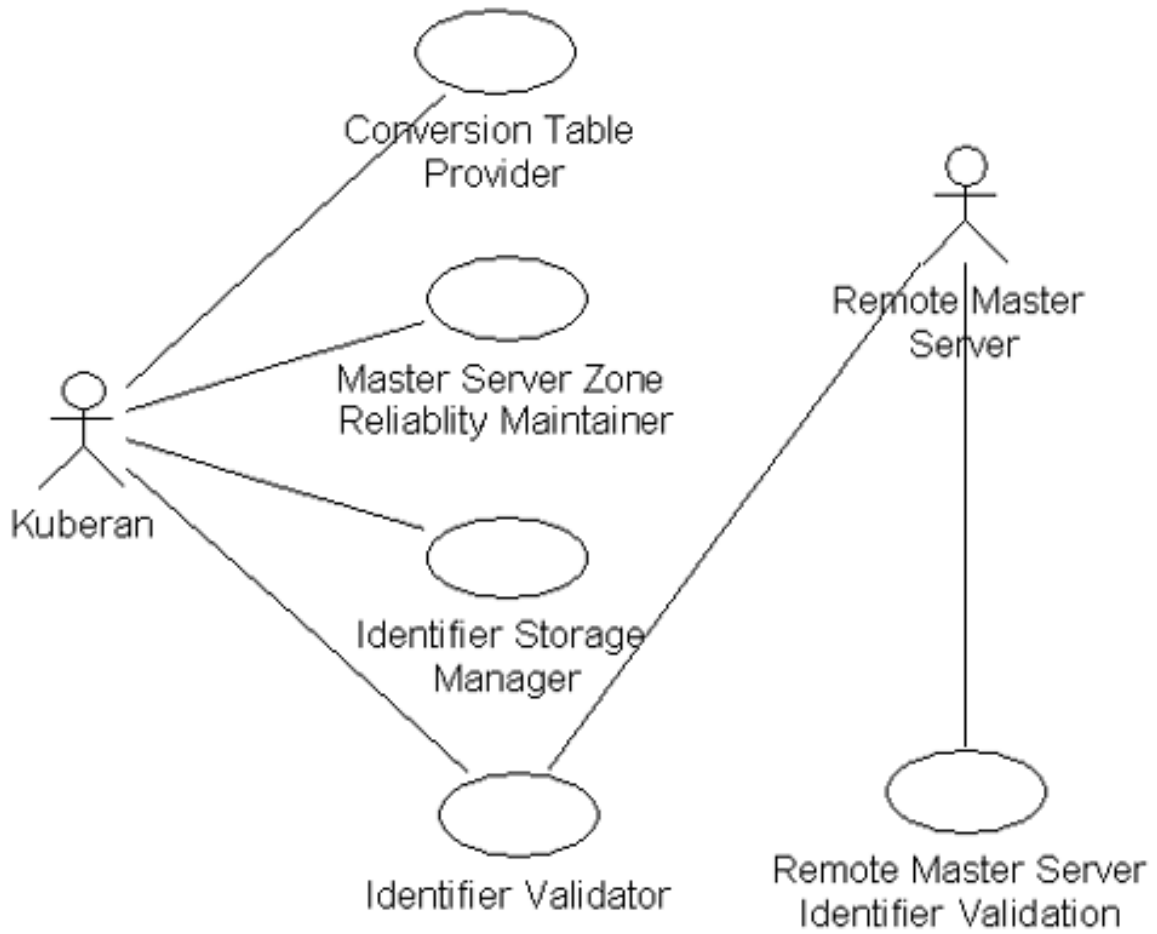


Figure 3.8: Brahma Use Case Diagram

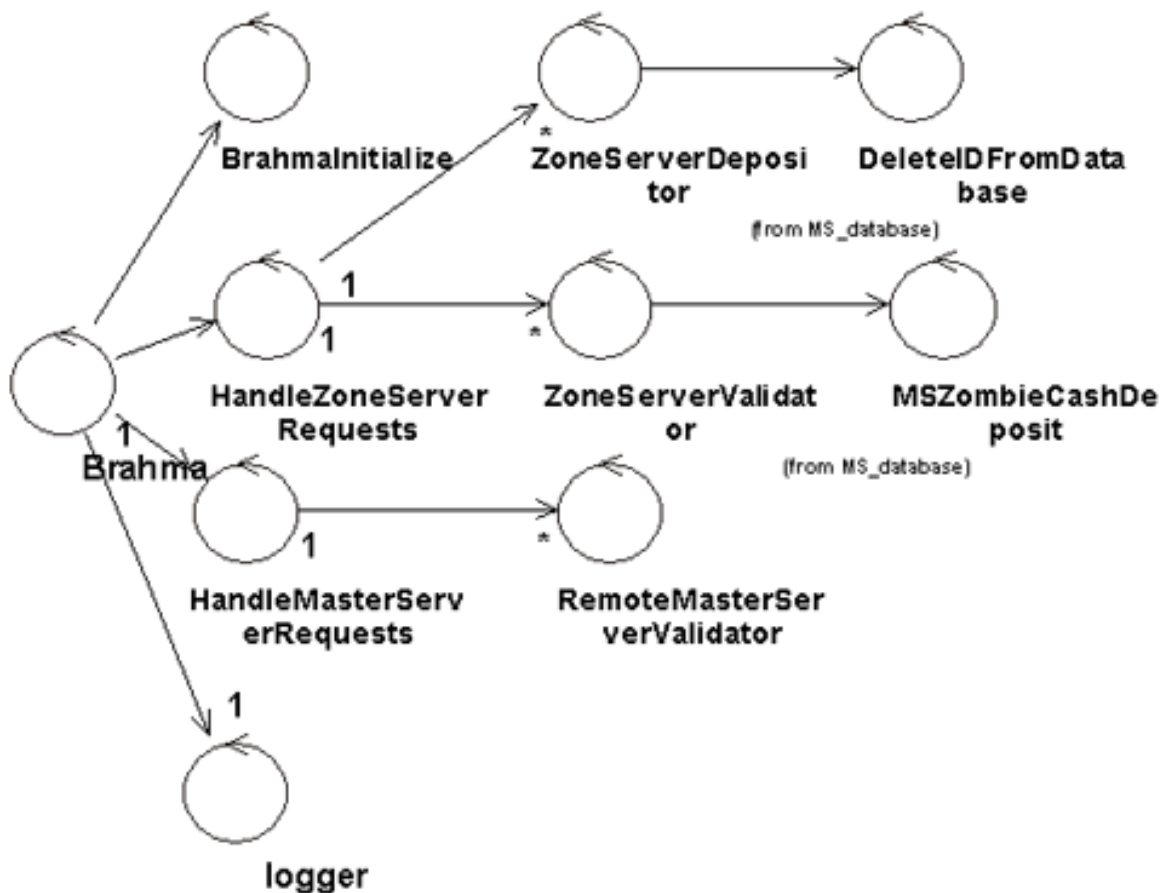


Figure 3.9: The Master Server Class diagram

The HandleZoneServerRequests class handles requests from the zone servers. This is basically handled by two threads. A ZoneServerDepositor instance is created when the zone server connects to a depositor socket. The second thread, the ZoneServerValidator thread, is instantiated when the zone server connects to a validator socket.

The HandleMasterServerRequest class is for remote master server requests that come to us as the identifier contains our MasterServer ID. These cases are verified at this location. Double spending is also handled here.

3.7 The DataBase Design

3.7.1 The Design of the Zone Server DataBase

The Zone Server Database needs to store information regarding users. For this three tables have been created. This is keeping in view the various security aspects and the normalization aspects (I have made a tradeoff on the side of security).

Table: 3.1 shows the password table. The field *frozen*, if enabled by setting it as 't', disables that user's login.

Table: 3.2 shows the Transaction table which is used to store the information about each transaction.

Table: 3.3 shows the Personal Information table, which is used to store the personal information of each user. Constraints have been set to allow modifications to this table on a constrained basis.

Name	Schema	Datatype	Size	Scale	Ref	Nulls?
USERNAME	<None>	VARCHAR2	255		✗	✓
PASSPHRASEHASH	<None>	CHAR	1		✗	✓
FROZEN	<None>	CHAR	1		✗	✓
PASSPHRASE	<None>	VARCHAR2	255		✗	✓

Table 3.1: The Zone Server Password Table

Name	Schema	Datatype	Size	Scale	Ref	Nulls?
USERNAME	<None>	VARCHAR2	255		✗	✓
CREDIT_DEBIT	<None>	CHAR	20		✗	✓
AMOUNT	<None>	NUMBER	0	0	✗	✓
CONNECTED_FROM_IP	<None>	VARCHAR2	255		✗	✓
TIMESTAMP	<None>	LONG			✗	✓
IDENTIFIER_USED	<None>	VARCHAR2	255		✗	✓

Table 3.2: The Zone Server Transaction Table

3.7.2 The Design of the Master Server Database

The Master server database has the responsibility of maintaining the Identifier information deposits and the hash, RAK. This is depicted in Table: 3.4. SETSTAT_CHK and SETVAL_CHK are set when STAT_CHK and VAL_REQ_MS reach the Master Server. The *scavenger* class removes entries from the database which have outlived their lifetime.

3.8 The Package Layout

The Logical arrangement of the system is shown in the diagram3.10 (p 20). This diagram shows how the classification has been done. The description is given below.

3.8.1 Main Packages

These are packages that contain classes that make the system perform in the manner expected by the PREDICT II.

brahma: This package contains all the control classes required for the Master Server software. This includes the database specific classes. The class design has been presented in Figure 3.9 (p 18).

kuberan: This package contains all the control classes required for the Zone Server. This includes the class diagrams shown in the Figure:3.7 (p 16).

kuchelan: This package contains all the packages required for the functioning of the user software in order to communicate with the remote client and the zone server. The class diagram have been shown in Figure: 3.5 (p 14).

Name	Schema	Datatype	Size	Scale	Ref	Nulls?
USERNAME	<None>	VARCHAR2	255		✗	✗
MONEYBALANCE	<None>	NUMBER	0	0	✗	✓
ADDRESS	<None>	VARCHAR2	300		✗	✓
MAILID	<None>	VARCHAR2	40		✗	✓

Table 3.3: The Zone Server Personal Information Table

Name	Schema	Datatype	Size	Scale	Ref	Nulls?
IDENTIFIER	<None>	VARCHAR2	255		✗	✗
AMOUNT	<None>	NUMBER	0	0	✗	✗
TIMESTAMP	<None>	NUMBER	0	0	✗	✗
RTT	<None>	NUMBER	0	0	✗	✗
RWT	<None>	NUMBER	0	0	✗	✗
RAK	<None>	VARCHAR2	255		✗	✓
K1	<None>	VARCHAR2	255		✗	✓
K2	<None>	VARCHAR2	255		✗	✓
HASH_K1_K2	<None>	VARCHAR2	255		✗	✓
SETSTAT_CHK	<None>	CHAR	10		✗	✓
SETVAL_CHK	<None>	CHAR	10		✗	✓

Table 3.4: TheMaster Server Identifier_Information table

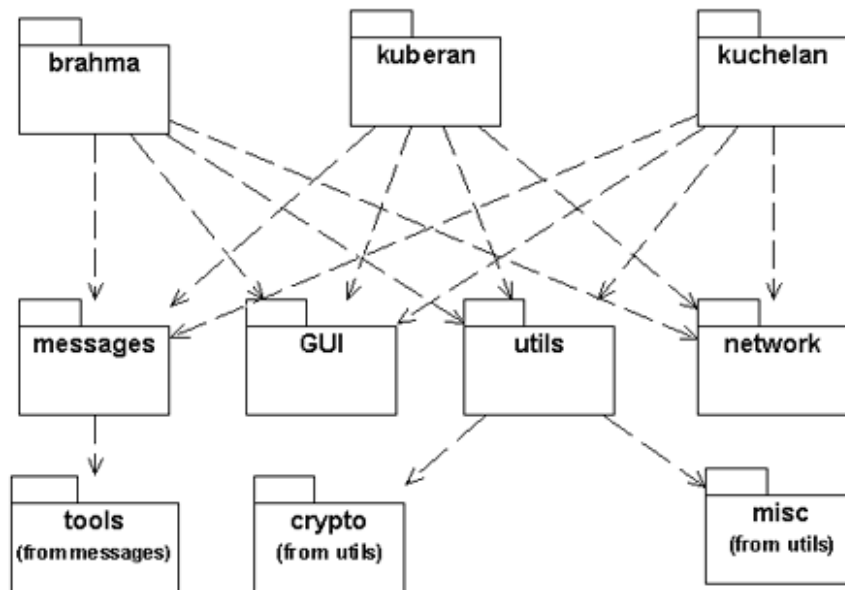


Figure 3.10: Package Layout of the project

3.8.2 The Helper Packages

These are the packages that store classes that are used for the common activities by the main classes.

messages: All the messages of the PREDICT II protocol have been embodied in classes of their own. The package that holds all these classes including the tools required for performing operations on them is stored in this package.

GUI: The Graphical User Interface classes required by any of the main classes are stored in this central repository.

utils: All operations related to file, encryption, filter operations, configuration file parsing are put together in this central repository for the use by all the main packages.

network: The network related classes such as the *networkAnalyserClient*, *networkAnalyserServer*, *SecureReader*, *Securewriter*, *InSecureReader*, *InSecureWriter* classes are stored here for use by all the main classes.

4 Implementation

The great aim of culture is the aim of setting ourselves to ascertain what perfection is and how to make it prevail.

- Mathew Arnold

4.1 Introduction

This chapter deals with the implementation of the system. This section describes how the artifacts of the design are mapped into executable code. Though it is not possible to specify in detail all the aspects of the system in this report, an attempt has been made to discuss the main issues involved.

4.2 The Working of the Client

The working of the client software has been depicted in the two state diagrams . The state diagram in Figure:4.1 (p 23) represents the steps involved in the withdrawal of a cash token from a zone server and sending the cash token to a remote client. The state diagram in Figure:4.2 (p 24) depicts the steps involved in requesting a cash token from a remote client and depositing the same with the zone server.

4.2.1 The Working of the Client as a Citizen

Invoked when the user decides to work in the citizen mode. The client then waits for remote client requests requesting for cash. On such requests, after the user sanctions, the client proceeds with the transaction. Figure:4.1 (p 23) shows the states of the system interactions.

The TRAN_ABRT message is transmitted when the requirement arises. The operations of sending REQ_REP and Authenticating itself with the zone server are done in parallel.

Generation of cash token information involves generation of a random number taking the system time as the generator's salt value.

4.2.2 The Working of the Client as a Merchant

Invoked when the user decides to work in the merchant mode. The client then connects to a remote client and requests for cash. On reply, after the user sanctions, the client proceeds with the transaction. Figure:4.2 (p 24) shows the different states of the system interactions.

4.3 The Working of the Zone Server

The Zone Server analyses the messages. The state of the system as the messages arrive from other entities is depicted in the diagram:4.3 (p 25). The abortion of any transaction involves sending the DEL_ID request to the master server *if* the identifier had been sent for deposit earlier.

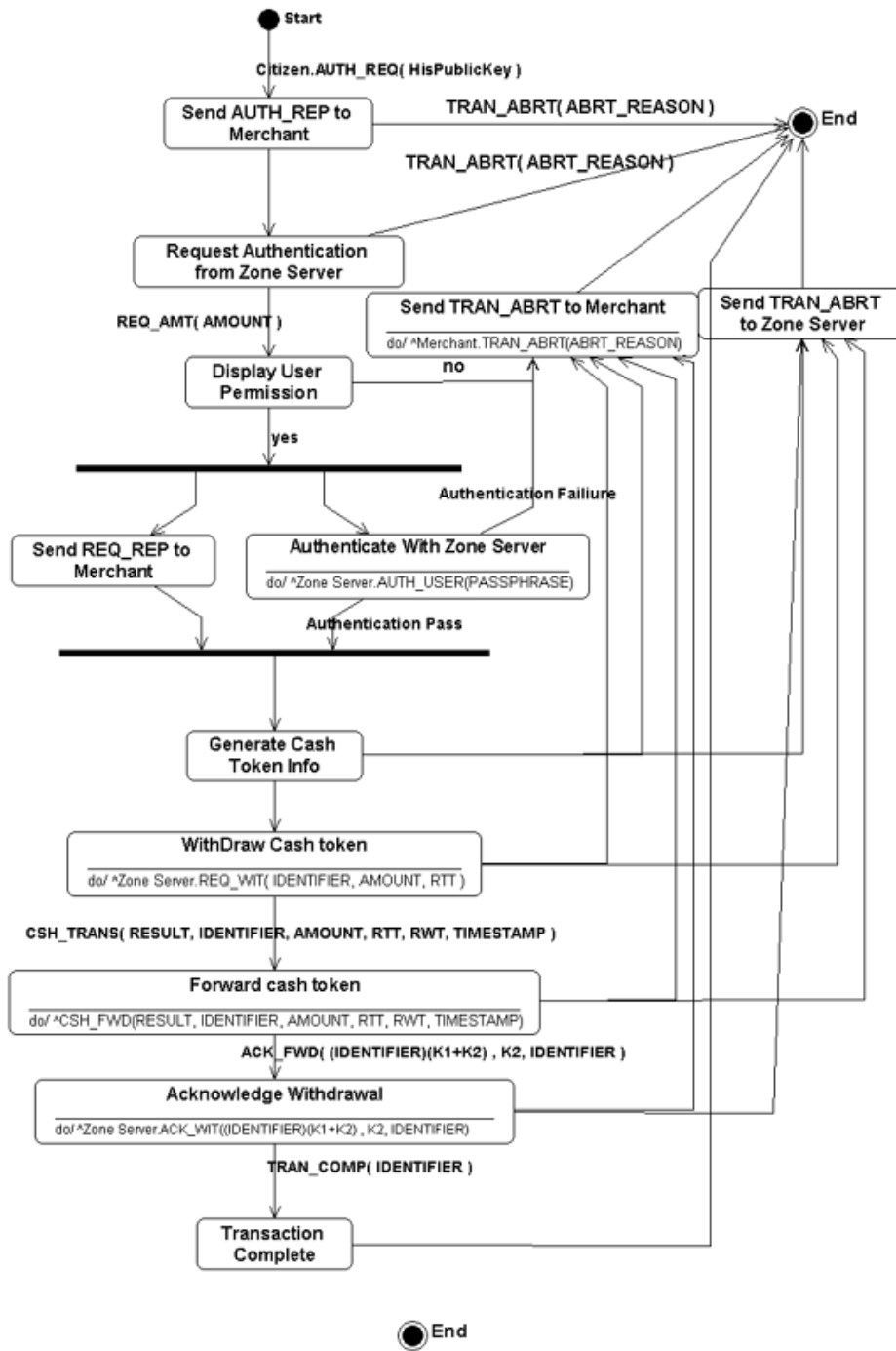


Figure 4.1: The state diagram for the Citizen part of the Client Software.

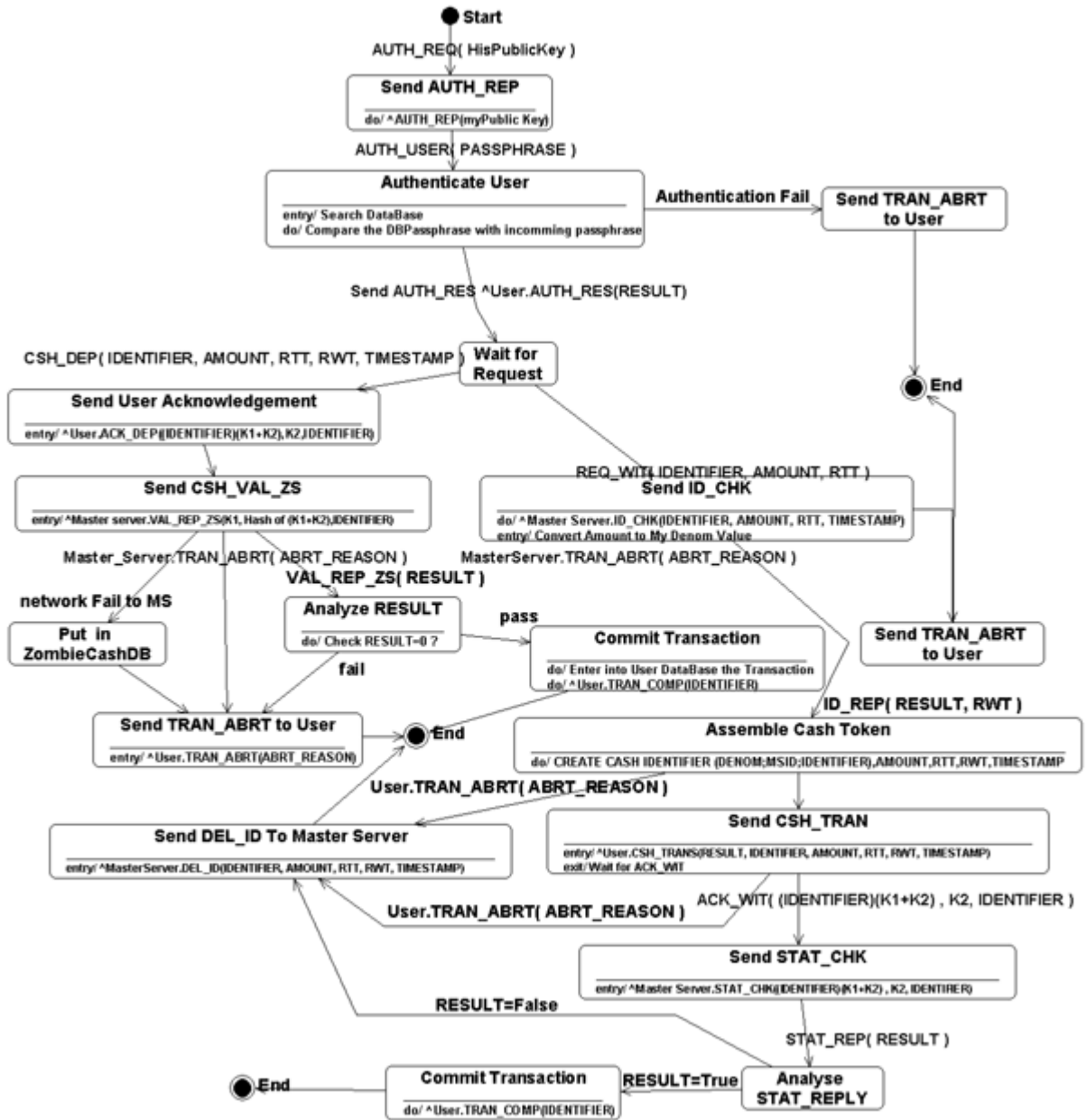


Figure 4.3: The Complete Zone Server Software State Diagram

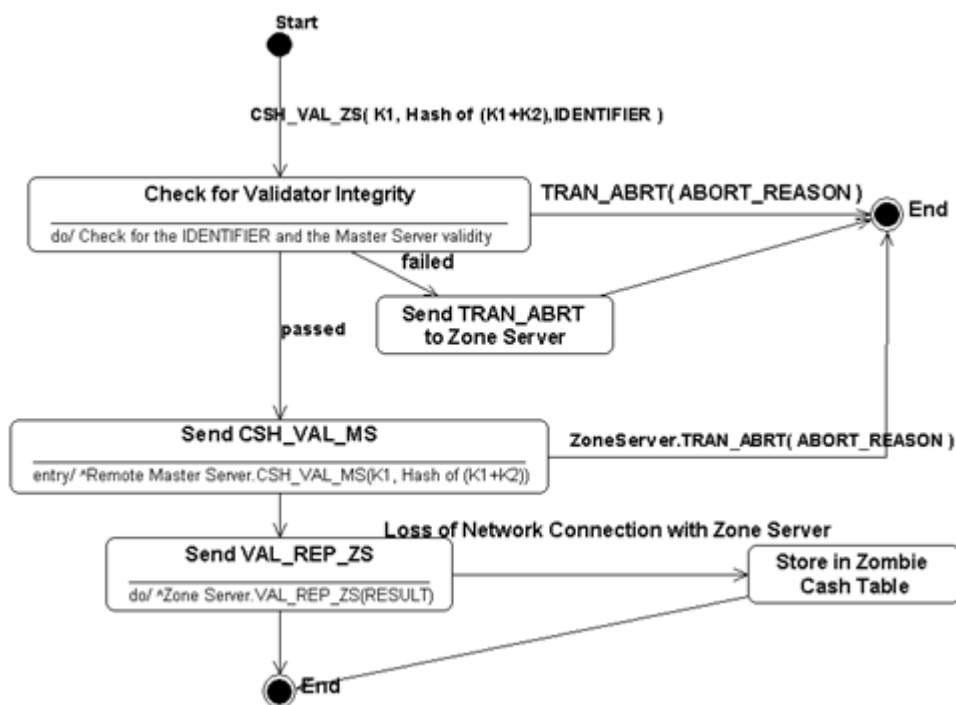


Figure 4.4: Master Server validator State diagram

A transaction is committed to the database only when it is sure that the transaction is a valid one and that it will be completed on the other side.

The Identifier is generated as per the specification for the format of an identifier as in Section 3.3.2.1 (p 10).

4.4 The Working of the Master Server

4.4.1 Zone Server Validator

The request from a zone server for deposit of cash is handled in the manner shown in the state diagram:4.4 (p 26). VAL_REP_ZS is send only when the database has been updated with RAK and checked out. However, if the request or the identifier is invalid in any respect, the TRAN_ABRT message is send. The wait time is measured as a factor of network delays and the computation time. If needed this is transmitted to the remote master server for validation.

4.4.2 Zone Server Depositor

The Zone Server depositor handles the deposit requests from the zone server and inserts the identifier into the Identifier_Information table of the brahma database. These set of operations are shown in the state diagram 4.5 (p 27). This occurs when the ID_CHK request comes in. ID_REP is send when the ID is confirmed as not being already present in the DB. If no duplicates are present then STAT_REP is send. In case of the ID being present, TRAN_ABRT is send.

4.4.3 Master Server Remote Validator

When a remote master server requests for the validation from our database it implies that one of the zoneservers had deposited the cash token and it is up to us to make sure that either the cash token is deemed valid or invlaid. This is shown in the state diagram 4.6 (p 27).

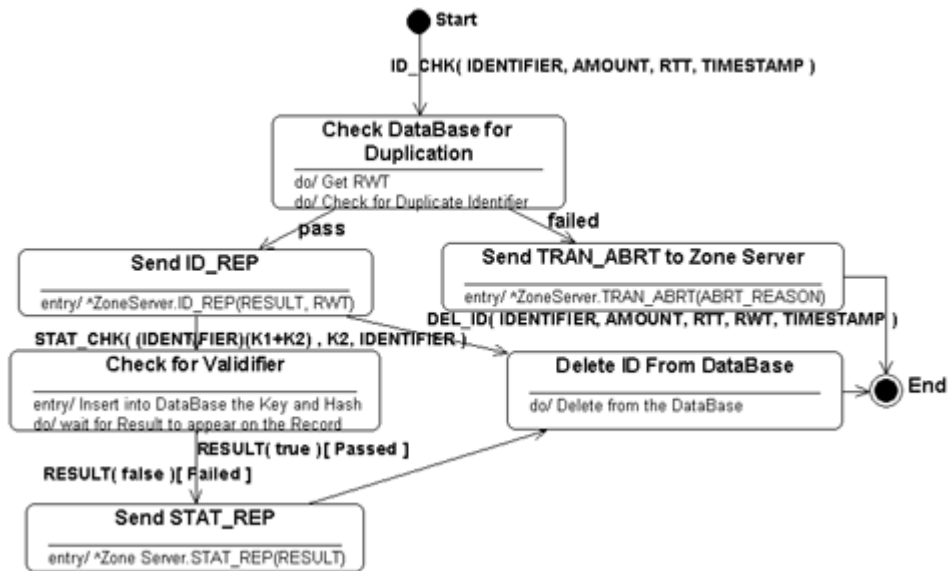


Figure 4.5: The Master Server Depositor State Diagram

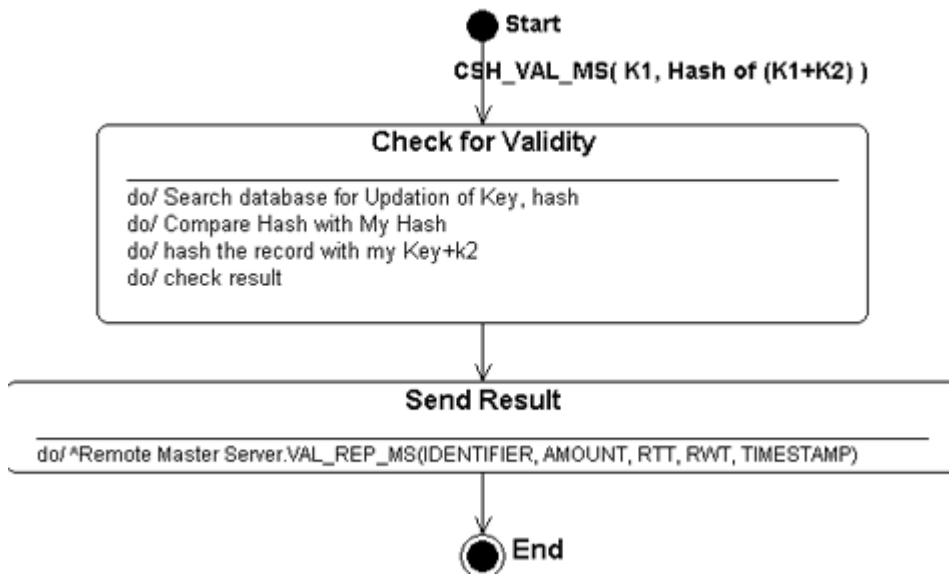


Figure 4.6: The Master Server Remote validator State Diagram

4.5 Highlights of the Implementation

This section deals with the major aspects of the implementation. Most of the implementation design is shown in the state chart and class diagrams in chapter3 (p 7). This section supplements that material for the reader with the innards of the system as such.

4.5.1 Asynchronous Network Transmission

Network packets have the nasty habit of arriving at times least expected. This was an expected problem that was handled by maintaining two separate threads for each socket. The first thread is for writing and has an access function that adds Strings to a buffer Vector, while the thread keeps on sending the data stored in the Vector one by one. The second thread is meant for reading data from the network; this thread reads the buffer and puts the data read into a buffer Vector, while an access function allows the parent classes to read from the buffer. This allowed for asynchronous transmissions and reception to take place.

4.5.2 Multi Threading

A fundamental concept in computer programming is the idea of handling more than one task at a time. Many programming problems in this project required that the program be able to stop what it was doing, deal with some other problem and return to the main process. Javas threading is built into the language, which makes a complicated subject much simpler. Threading is supported on an object level, so one thread of execution is represented by one object. Java also provides limited resource locking. It can lock the memory of any object (which is, after all, one kind of shared resource) so that only one thread can use it at a time. This is accomplished with the *synchronized* keyword. This has indeed been a boon for the management of the system.

An inherent problem that had to be handled in this project has been to tell a networking thread when to stop listening to the network and shut down the network properly and die. This is needed especially when the TRAN_ABRT message is received or when a failure is intimated by some message (TRANS_COMP , CSH_TRAN etc.) . This turned out to be the bane of the system. This problem was solved by the deprecated method *stop(Exception e)*. Once a flag is set in the thread, the thread will never check for the *boolean* as it is blocked while waiting for a read from the network. *StopThreadException* is thrown into the thread so that it quits. This however results in the Exception leaking out at times and the exception gets thrown when the thread is dead (The timing of the controlled throwing of the Exception by the control class may not be always correct). So the parent class has to handle the Exception which it is trying to throw, else the parent class may behave strangely.

4.5.3 Data Base Concurrency

This problem is handled by the following methods.

- Maintaining separate connections for each of the SQL queries to be handled. The *Kuberan-Database* and the *Database* classes handle the database activities. Concurrency to the access methods is maintained by the use of *synchronized* modifier of Java, which forces blocking of calling threads until the thread currently executing the method is completed.
- The database maintains its own concurrency when more than one process connects to it. Oracle is exemplary in this aspect.

4.5.4 Time Analysis

4.5.4.1 Introduction

Analysis of time is one of the most complicated activities that can ever be done on a network. However the analysis is possible within a certain probability by the grace of mathematics. The computers send their own times a finite number of times. This topic has not been covered as this, though being important for the system, is the responsibility of a layer not in the system. Since this layer was absent in my platform, I had to code it in.

4.5.4.2 The Working

The remote computer first sends us his time (T_1). In return we send him our time (T_2) when we received the message. On receipt of this, the remote computer transmits T_3 . Say, this reaches us at our time T_4 . The following computation is done.

Difference in Time between System times of the two computers = $T_2 - (T_1 - \text{network delay})$

Network delay = $\frac{T_4 - T_1}{2}$

Thus,

The difference in the System times of the two computers = $T_2 - (T_1 - \frac{T_4 - T_1}{2})$

Over n transmissions, the estimate of the Network delay

= $\frac{\sum \text{Network Delay In Each Transmission}}{n}$

Similarly,

The estimate of the System Time difference is also computed.

4.5.4.3 The Implementation

The implementation is done by having a *networkAnalyserClient* class and a *networkAnalyserServer* class that uses a *DataInputStream* and a *DataOutputStream* as input and using the same to send 30 statistically viable transmissions. Using these values, each end computes the network times. The difference in time is also estimated as the *SuperMessage* class may also contain information as to the time when the data was send.

5 Testing

The progress of the rivers to the ocean is not so rapid as that of man to error.
-Voltaire

5.1 Introduction

The system after the implementation is a set of code with no guarantee to work and at times may need major rework if any serious failures have been detected. My approach towards this aspect has been to create a plan of action at the very start of the implementation phase and manage the development process so that there is the least amount of rework needed later in the development process. This chapter speaks about how this was done and what the results were.

5.2 Test Plan

The main plan that I followed was on the basic fact that all Java classes allow for the main operation to be present. This allows to create self testing code. Every class that uses or extends or implements the tested code considers that class to be perfect. Thus the responsibility of making the checks reside solely with the main operation of each class. At times, it would not be possible to test the class alone as it might depend on another class. These situations were handled by testing the involved classes together. In such cases the main operation of the involved classes had the responsibility of the integrity of the classes.

5.3 Unit Testing

Each of the Units of the class – especially the helper classes were tested and retested over and over again to make sure that most of their structure had been tested. Some of the classes such as the classes in the *network* package, *networkAnalyserClient* and the *networkAnalyserClient* were tested together so that these integrate perfectly.

5.4 Integration Testing

The project had many threads running together, coupled with data transmissions across these threads and the classes. It was thus imperative to test these to the fullest in order that they work. Many stress tests were launched and the logic tested against many set of situation.

5.5 Validation Testing

At the culmination of the integration testing, the software was completely assembled as packages, interfacing errors were uncovered and corrected, and a final series of software validation testing was

done. Here the system functions were tested in a manner that can be reasonably expected by a user. The system was tested against the requirement specification.

5.6 Output Testing

After performing validation testing, the next phase in testing was the output test of the system, since no system could be useful if it does not produce the desired output in desired format.

5.7 Conclusion

While most tests passed through, some of the tests failed miserably such as connecting more than 59 connections to the Oracle 8i server, number of threads restricted to 26% of memory. Overall, the restrictions of the extraneous system were generally the reasons for the failures, while some of the logical errors were also caught in each of the phases. Number of errors decreased as the testing proceeded and with an elaborate test involving 6 computers the system performed in a satisfactory manner.

Overall the results were acceptable.

6 Conclusion

It is better to light one candle than curse the darkness.
– Chinese proverb

6.1 Introduction

The system presented has its own set of failures and successes. However, the radical thought that I have suggested and tried out in my project will find very few takers. This is because a definite amount of fidelity still remains in the human race to the traditional manner of doing business. But then, the future will see EPS as the definite torch bearer of today's monetary system. And it may even be possible that PREDICT be considered as one of the steps in that direction.

We move from Chapter 2 where we saw the foresight about doing the project, to actually doing the project in the last two Chapters. Thus logically following the flow of events, this Chapter looks back at the project with a hindsight.

6.2 Limitations of the system

- **Performance.** Using Java on the server side caused a severe performance loss on the computations.
- **Time analysis.** Analysis done here is rudimentary in nature and hardly reflects the true network and computational complexities at most times.
- **Heavy dependence on Time Factor.** The dependence on a factor that is inherently variant in nature makes it difficult to predict the performance of the system.
- **Network utilization.** The network bandwidth usage is very high compared to most EPS.
- **User Intuition.** Using IPs for identification of remote clients makes it difficult for naive users to be at ease with the system.
- **Complexity.** The system as a whole is complex in nature and any implementation will require very large investment on the testing for perfection.
- **Vulnerable to DOS attacks.** Launching crippling Denial of Service attacks against the zone servers is childishly simple and involves about 100 lines of java code.

6.3 Advantages of the system

- **Resilience against obsolescence.** Ability of the system not to be dependent on a single encryption system, but a plethora of candidate systems makes the system solid against becoming obsolete with technology.

- **Scalability.** The master servers can easily supplement zone servers with more numbers as the user load on the zone increases.
- **Ease of Use.** Compared to most EPS systems, with the minimal technical knowhow about the system, it is possible to do business without bothering about the cyber wallets, my money timing out etc.. It is either complete transaction or none at all, but the money is always safe.
- **Security issues related to EPS solved.** Problems such as double spending have been solved in the usual manner making sure only one of the transactions take place.
- **Very Secure implementation compared to most EPS implementations.** The use of session key, strong encryption for data transmission and creation of a DMZ zone between the web server and the Database server was some of the innovative ideas that secure the system.

6.4 Future Work

- Time analysis should make use of the statistical data available. Thus a better predictive algorithm can be developed
- Transaction cost to be measured.
- Analysis of official work involved, their social implications and the costs have to be analysed.
- Recoding both the server modules into C++.
- Code profiling and subsequently optimization to be done.
- Testing on a representative network other than a Local Area Network.

Bibliography

- [1] Geoffrey Crowther, 1994, *An Outline of Money*, Universal Book Stall, 5 Ansari Road, New Delhi
- [2] Pete Loshin and Paul A. Murphy, 1998, *Electronic Commerce*, Charles River Media, P.O. Box 417, 403 VFW Drive, Rockland, Mass.
- [3] Kolkata R and Whinston A, 1995, *Frontiers of Electronic Commerce*, Addison-Wesley Publishing Company Inc., Mass.
- [4] Martin Fowler and Kendal Scott, 2000, *UML Distilled*, Addison-Wesley Publishing Company Inc., Mass.
- [5] Grady Booch, James Rumbaugh and Ivar Jacobson, 1999, *The Unified Modelling Language User's Guide*, Addison-Wesley Publishing Company Inc., Mass.
- [6] Ken Arnold and James Gosling, 1998, *The Java Programming Language*, Second Edition, Addison-Wesley Publishing Company Inc., Mass.
- [7] Bruce Eckel, 1998, *Thinking in Java*, Addison-Wesley Publishing Company Inc., Mass.
- [8] Naughton P, 1999, *The Complete Reference to Java 2*, Tata McGraw-Hill Publishing Company Limited.
- [9] <http://espo.jrc.es/paysys.html>
- [10] <http://www.mastercard.com/set/set.html>
- [11] <http://www.digicash.com>
- [12] <http://www.cybercash.com>
- [13] <http://pdf.computer.org/books/ic/1997/pdf/w6004.pdf>
- [14] <http://www.openssl.org>
- [15] <http://www.gismo.net>