

# Distributed Gaming using XML

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment of the Requirement for the Degree

Master of Science

By

Padmini Paladugu

June 2004

© June 2004

**Padmini Paladugu**

paddusyam@yahoo.com

**ALL RIGHTS RESERVED**

**APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE**

# Abstract

Advancements in wireless technology have led to the emergence of a wide variety of wireless devices like Personal Digital Assistants (PDAs) and cell phones. A number of applications like word processors have been developed to run on these devices and the most popular among these are gaming applications. In this project, we have developed a Pokemon-style game, *Palm Maya*, played on wireless devices. *Palm Maya* uses a stripped down XML language that we had created to communicate with a centralized Oracle Database. Various properties of our XML language and game set-up have been tested for efficiency and playability.

The basic set up of *Palm Maya* is that we have a number of wireless users on one hand and a centralized Oracle database on the other. Each player starts the Pokemon-style game and chooses a level of difficulty – Beginner, Intermediate, or Expert (an option about what kind of player he/she will be in the game). The player's choice will then be recorded into the centralized Oracle database. If two persons decide to play each other, the system decides who begins the game. The players can point their devices at each other and beam their transactions to each other. Also, players can synchronize their devices with the centralized Oracle database to update their scores. The centralized database has a high score list which can be synchronized with the palm devices. The centralized Oracle database keeps track of all XML based messages that have passed between the devices. Communication between players was achieved using Infrared (IR) beaming and Bluetooth technologies.

## TABLE OF CONTENTS

<b>1. INTRODUCTION</b>	1
<b>2. BACKGROUND AND RELATED WORK</b>	4
2.1 Palm OS Concepts	4
2.2 XML	6
2.3 SAX Parser	7
2.4 Oracle XML Database	7
<b>3. REQUIREMENT ANALYSIS</b>	9
3.1 Purpose	9
3.2 Scope	9
3.3 Benefits and Objectives	11
3.4 Product Perspective	11
3.5 User Classes and Characteristics	13
3.6 Operating Environment	13
3.7 External Interface Requirements	13
3.8 Functional Requirements	14
<b>4. DESIGN AND IMPLEMENTATION</b>	19
4.1 System Architecture	19
4.2 Design	22
4.3 Implementation	35
<b>5. USABILITY TESTING</b>	
5.1 Usability Testing	60
5.2 Results	61
<b>6. CONCLUSION AND FUTURE WORK</b>	64
<b>7. REFERENCES</b>	66

## LIST OF FIGURES

Figure 1: Distributed Game System .....	12
Figure 2: Three-tier Distribution Architectural Pattern .....	20
Figure 3: System Architecture .....	21
Figure 4: Game Server .....	23
Figure 5: Initial Form .....	27
Figure 6: Class diagram for Server .....	39
Figure 7: TossForm and MainForms in the Game .....	42
Figure 8: State diagram representing the game .....	43
Figure 9: Previous Play state .....	47
Figure 10: Current Draw state .....	47
Figure 11: Attacking state .....	49
Figure 12: Attacking state .....	50
Figure 13: Application Preference .....	54
Figure 14: Quit the Game .....	55
Figure 15: Game Rules .....	55

## LIST OF TABLES

Table 1: Operating Environment .....	13
Table 2: Trading Conditions .....	30
Table 3: Structs for Deck and Player's Details .....	56
Table 4: Usability Testing Results .....	63

# Chapter 1

## Introduction

New classes of devices like wireless-enabled PDAs and mobile phones are emerging. Mobile and multi-player gaming applications on these wireless-enabled devices are becoming increasingly popular. Initially, the gaming market has been dominated by single player and multi-player games that are played on PC and Game Boy families. Currently, mobile games can be unlike any of them as the devices are limited in terms of media, but networked and multiplayer compatible. The advent of distributed games has opened up new avenues of entertainment for users.

The general problems associated with designing distributed games on PDAs are providing communication between game players, maintaining consistent state between players and central database, and managing all players' data in the database simultaneously. To facilitate distributed gaming, technologies like Palm OS, Bluetooth, eXtensible Markup Language (XML), and Oracle are widely used.

The Palm OS is the most common OS for PDAs. It provides both simple applications like date book, calendar, and notepad, and more advanced applications like games to handheld users. Users can add more advanced features to existing applications or they can write their own applications using the features of Palm OS. The Palm OS allows sharing of data between the Palm devices using Infrared and Bluetooth communication.

Palm networking allows the users to connect to the remote database using Palm OS Net library. Palm OS supports communication between handheld and desktop computer through Palm cradle. The Hotsync manager allows the user to make backup copies of his/her handheld information. Palm Conduits allow the actual synchronization between the applications on the Palm device and the Palm desktop. Data interchange between the Palm devices and the Palm desktop on a host computer is best handled by XML messages.

The eXtensible Markup Language (XML) is designed for describing the structure of information, which makes it easier to transfer ordered information from one place to another place, or from one program to another program. It provides the syntax for defining the structured information using XML Schemas or Document Type Definitions (DTDs). The XPath and XQuery features of XML can be used to store and retrieve XML messages from a relational database system like an Oracle Database.

Oracle9i provides support for handling XML data and documents. Oracle 9i provides a data type called XMLType that can be used in defining tables, columns in tables and views. Oracle9i allows XPath expressions to navigate through an XMLType instance and allows searching across multiple instances of XMLType.

This project develops a distributed gaming system where multiple players with Palm PDAs can play an interactive Pokemon-style game. Palm OS graphics and other user interface elements are used for creating an interactive game interface. Palm Databases

and Preferences are used for storing player's details and game components. Palm OS Memory Manager is used to handle the memory handling operations. Palm OS Net Library is used for communication between Palm PDA and remote database at the remote Server. Bluetooth and Infrared technologies are used for providing communication between wireless devices, and the devices communicate with the centralized Oracle database using XML messages.

This report is organized as follows. Chapter 2 provides the background and related work on Palm OS, Bluetooth, Conduit Development, XML, XML Parsers, and Oracle XML Database. Chapter 3 explains the requirements analysis for the project. Chapter 4 describes the Design and Implementation details of the project. Chapter 5 discusses the usability testing that was conducted to test the validity of the game and the results. Chapter 6 concludes this report and gives directions for future research.



# Chapter 2

## Background and Related Work

This chapter discusses the technologies used in this project to implement a distributed gaming system. The concepts of Palm OS, Conduit Development, XML, XML Parsers, and Oracle XML Database are discussed below.

### 2.1 Palm OS Concepts

Personal Digital Assistants (PDAs) are compact and portable. They do not have a hard disk to store programs and applications and have only a small amount of Random Access Memory (RAM). These limitations of handheld devices require a special operating system such as Palm OS. Palm OS is the predominantly used technology in the field of handheld devices and is developed by PalmSource, Inc. Palm OS technology broadly includes communication technologies used in the devices such as PDAs, Cell Phones, and Pagers. Since the initial release of Palm devices, PDAs have been a part of the gaming industry.

#### 2.1.1 Palm OS Concepts used in our Project

The important Palm OS concepts used in game application are the following: User Interface elements like Forms, Lists, Text fields, Buttons, Menus; Palm OS databases; Bluetooth and IR Beaming technologies; Palm OS Networking; Application Preferences; and Palm Conduit.

**Palm User Interface:** The Palm user interface is different from the other user interfaces due to its small size and behavior. Users can view only one application at a time and the application's forms cannot be resized or moved. The main user interface resources we used in the project are Forms, Lists, Buttons, Fields, Labels, Alerts, and Menus.

**Palm Databases:** Palm database is basically a list of memory chunks in the Palm's storage RAM, along with some header information that describes the database itself. Databases are opened, closed, created, and deleted just as files on the other traditional file systems. The Palm Data Manager organizes all the records associated with a Palm database and provides functions that allow users to create, query, update and delete those records. In this project, the Palm database is used to store a deck of thirty cards required for the game, and the player's details.

**Palm Memory Handling:** The Memory allocation on a Palm is limited to slightly less than 64KB for each allocation and these memory allocations are called *chunks*. The Memory Manager returns either a memory handle or a memory pointer for each memory allocation. A *memory handle* is a reference to a movable block of memory. A *memory pointer* is a reference to a non-movable block of memory.

**Palm Application Preferences:** The Palm OS does not support the execution of multiple applications simultaneously. The application shuts down when a user switches to another application. The Application will not save the most recent state where the user had last worked. Losing state causes problems with many applications. To avoid this ambiguity in

the state of the applications, Palm OS provides application preferences. The application's state is stored in the *Preferences* database when a user exits the application and it is retrieved when the application is re-launched. In this project, application preferences are used to maintain the game state.

**Palm OS Exchange Manager:** The Exchange Manager provides communication between palm devices via Object Exchange (OBEX) protocol. Although mostly associated with infrared, the OBEX protocol can also run over Bluetooth, TCP/IP or even serial communication. OBEX transmits objects as streams of octets. The content of the transfer might contain a single record, multiple records or even a complete database. In this project, Bluetooth and Infrared beaming technologies are used for communication between players.

**Palm Conduits:** To synchronize the data on the Palm device with a host computer, a software interface must be provided to handle the synchronization from the desktop computer's perspective. This software interface is called a *conduit* because it manages the flow of data through a pipe between a desktop application and a handheld application.

In this project, the Palm conduit is used to synchronize the deck of cards to the Palm database and player's score to the centralized database. Conduits work in tandem with the *HotSync Manager* to perform the following tasks:

- Mirroring and synchronizing with data on the other systems
- Backing up the data residing on the Palm
- Downloading and installing Palm OS applications

- Importing and Exporting data

## **2.2 eXtensible Markup Language (XML)**

XML belongs to the family of markup languages. It is similar to Hyper Text Markup Language (HTML). XML provides features for user-defined tags. It also provides features for checking the wellformedness and validity of the data described. Validity check of XML documents is required to make sure that a given XML document follows the defined rules. Each XML document is validated against its corresponding XML Schema. If the document satisfies all the constraints specified by the schema, it is considered to be schema-valid.

## **2.3 SAX Parser**

*Parsing* involves reading an XML document and retrieving its content while checking for the document's wellformedness. SAX is an event based API used for reading XML documents. SAX parser is used to create an XML-to-Java mapping for both simple and complex XML structures. SAX is a collection of interfaces in the *org.xml.sax package*. As SAX is an API, the code is standard across all XML parsers. To run Java applications using SAX, we first need to access an XML parser that supports SAXv2. We have used Apache's *Xerces* parser to support SAX and DOM.

## **2.4 Oracle XML Database**

The XML features of the Oracle9i Database Management System (Oracle XML) provide tools for building XML applications. Oracle XML can be used to store, query, update,

transform, and process XML documents, while providing Structured Query Language (SQL) access to the XML data. The key Features in Oracle XML are XML Types, XML schema, XML Schema validation, and XPath search. *XMLType* datatype can be used to store and retrieve XML data from the Oracle database. *XML Schema validation* in Oracle XML allows validation of XML documents stored in Oracle database against their schema. *XPath search* uses XPath syntax to query XML content in the database.

### **2.4.1 Store XML data**

XML documents are stored in the Oracle database either by using an XMLType column or an XMLType table. To store an XML document in an XMLType table or column, the XML document must first be converted to an XMLType instance. This is done using the `getDocument ()` PL/SQL function. After the conversion, the XML document will be stored in the XMLType table using the SQL INSERT statement.

### **2.4.2 Retrieve XML data**

After storing the collection of XML documents into an XMLType table or column, the next step is retrieving the XML documents. Oracle 9i database uses the concept of XPath to traverse through all the sub elements from the root element. Oracle XML uses the XPath in conjunction with the *extract ()*, *extractValue ()*, and *existsNode ()* functions. The *existsNode ()* function verifies the presence of a node in the XML document. The *extractValue ()* function is used to retrieve the value of a text node or an attribute associated with an XPath Expression. The *extract ()* function is used to retrieve all the nodes in an element node specified in the XPath expression.

### **2.4.3 Update XML data**

XML documents in the database can be updated using the *updateXML ()* function. The *updateXML ()* function can update the value of an attribute, node, text node, or node tree. The target for the update operation is identified using an *XPath* expression.

# Chapter 3

## Requirement Analysis

This chapter discusses the requirements for this project.

### 3.1 Purpose

The purpose of this project is to provide a distributed, multiplayer gaming application for Palm OS enabled wireless PDAs. The main idea behind this project is implementing a distributed game that connects to the centralized database by integrating diverse platforms and technologies.

### 3.2 Scope

The scope of this project is implementing a more sophisticated game, *Palm Maya*, on wireless devices using Palm OS, Bluetooth, XML and Oracle 9i XML Database. *Palm Maya* is a modern remake of the classic card board game *Magic: The Gathering*. It is a simple card trading game, scaleable to varying levels of difficulty. The following section describes the overview of the game and the different phases being implemented in the game.

#### Game Overview

The *Palm Maya* play takes place between at least two players on two different Palm devices. There are two types of cards: Lands and Animals. Land cards are used to pay for the Animal cards. The cost of a Land card is always one unit. Each Animal card has an image with three properties: cost, power, and toughness.

- Cost: The cost of the Animal card will decide how many Land cards we need to pay for playing the Animal card.
- Power: Power is the how much damage the players deal in combat.
- Toughness: Toughness is how much damage a player takes to destroy the Animal.

### **Rules of the Game**

Each player has a deck of thirty cards in his/her Palm database. The player launches the application with an initial set of seven cards in his/her hand and plays the game according to the following predefined rules.

1. Each turn the player needs to draw a card from the deck.
2. The player may play one land card during a turn.
3. The player can play more than one animal card as long as he has sufficient number of land cards to pay for them.
4. The player need not pay again for the animal card, which has already been played in the previous turn.

### **Game Play**

There are four different phases in the game: Draw, Play, Combat and Update score. First, each player draws a card form the deck, and then plays either a Land card or Animal card from the hand list. If the player is qualified to attack the opponent, then the combat phase of the game starts. Depending on the characteristics of the player's attacking Animal, the opponent takes the decision whether to block the attacking Animal or not. If the opponent blocks the attacking Animal, then a trade will occur between the players depending on the characteristics of both the attacking and blocking cards. If the opponent does not block the attacking Animal, then the player's score will be updated



according to his Animal's power. The main objective of every player is to be the top scorer among all the players currently playing the game. The game application should always maintain the scores of game players in the centralized Oracle database. Players can always have the choice of viewing top five scores from the Oracle database.

### **3.4 Product Perspective**

The end product of this project is integration of Palm devices, Conduit application, Game Server, and Database. Game Server deals with generating decks and updating scores of the players. Conduit provides communication between the Palm PDA and game server on the desktop. Oracle Database stores all the versions of the decks used by the players in the game. Palm device is the place where the player can play the actual game. The following Figure 1 depicts the processes contained in the Distributed Game system.

<b>Distributed Game System</b>			
1. Game on Palm device	2. Conduit	3. Game Server	4. Oracle XML Database
1.1 Launch the game application	2.1 Retrieves the data from the desktop file.	3.1 Receives the request from the client/player.	4.1 Maintain decks and scores of all the players with the Time stamp
1.2 Decide who goes first	2.2 Convert it to Palm Record format.	3.2 Decides the type of player.	
1.3 Start the game with Draw state	2.3 Store it to the Palm database via hotsync.	3.3 check in the database for existing player.	
1.4 Play Land cards or Animal cards.	2.4 Retrieve the data from the Palm Database	3.4 Create new deck with version update for existing player	
1.5 Attack or block the opponent.	2.5 Convert the data to the Desktop record format.	3.5 Create a new deck with new version for a new player.	
1.6 update the scores	2.6 Write to the Desktop file.	3.6 update the score for existing player.	
1.7 Quit the game.			
1.8 update scores to central database through Hotsync and conduit.			

Figure 1: Distributed Game System

### **3.5 User classes and characteristics**

While designing the game it is assumed that the users will have varying levels of education and technical expertise with Palm PDAs. Users should also become familiar with how to use a built-in keyboard and Graffiti area for providing input to a Palm device, and how to read the Palm PDA's screen to analyze the output. Users should also know how to install a program on Palm device, and how to launch a program on a system that uses a Graphics User Interface.

## 3.6 Operating Environment

The distributed game is required to operate on Palm devices with a centralized Oracle database. The hardware platform, operating system, and other software components are described in Table 1.

Table 1: Operating Environment

Application	Operating Environment
Game	Cygwin, Palm SDK and PRC tools on MS Windows. (Palm OS)
Conduit	Visual Studio, Conduit Development Kit (CDK) , Palm Desktop, and Hotsync Manager on MS Windows.
Game Server	Java 1.4.1 or higher installed on Windows.
Database	Oracle 9i

Source: Author

## 3.7 External Interface Requirements

### 3.7.1 User Interfaces

Players will take an average of five minutes to become familiar with the user interface and Palm device functions. For experienced Palm users, the average time will be one minute to learn the user interface. Experienced users are those who have previously interacted with a Palm application.

### 3.7.2 Hardware interfaces

The hardware interface for the *Distributed Gaming System* will be a Palm device, Palm Cradle, standard keyboard, mouse and monitor. The system will also require a mobile phone as a modem to interact with the Game Server database over the Internet.

### 3.7.3 Software interfaces

- Game: The operating system must have *cygwin*, *Palm SDK*, *PilRC*, *Palm Desktop*, and *Hotsync Manager* installed.
- Game Server: The operating system must have the Java Virtual Machine (JVM) version 1.4.1 or greater installed.
- Database: XML supported Oracle database must have been installed on top of the operating system.

### 3.7.4 Communication interfaces

- Communication between the Palm application and the central database will use Palm Network Library.
- Communication between two Palm devices will use Bluetooth and/or Infrared beaming technologies.

## 3.8 Functional Requirements

The functional requirements of the Game, Game Server, Conduit, and Centralized database are explained in this section.

### 3.8.1 Functional Requirements for Game

*Palm Maya* game should be able to perform the following functions.

**Toss:** This function occurs by default when the game application is launched. This function decides which player will start the game first.

*Input:* The input for this function will be the random number generated by the server.

*Output:* When the game application is first launched, the first form that will appear on the application window is the Toss Form.

**Deck:** This function occurs by default when the game application is first synchronized with the handheld. A deck of thirty cards will be required for playing the game. The deck of thirty cards will be available in the Palm Database.

*Input:* The input for this function will be the deck created by the game server.

*Output:* The deck of thirty cards stored in the Palm database.

**Handset:** This function occurs by default when the Main form is first launched after the Toss form was launched. Initially, handset is the set of seven cards for each player. The player can add cards to the handset in the draw state and remove cards from the handset in the play state.

*Input:* The input for this function will be the deck in the palm database.

*Output:* The output is first seven cards from the database displayed in the List in the Main form.

**Draw:** This function occurs when the user draws a card from the database. During the draw stage, either Animal or Land card will be retrieved from the database.

*Input:* The input for this function will be the deck in the Palm database.

*Output:* The output is new card added to the handset.

**Play:** This function occurs when the user plays a card from the Handset.

*Input:* The input for this function will be Handset.

*Output:* Played cards will be displayed on the screen.

**Beam:** This function occurs when the user is ready to transfer his turn to the opponent player.

*Input:* The input for this function will be player's response or player's data.

*Output:* The output will be sending data to the opponent player.

**Attack:** This occurs when the player attacks the opponent player.

*Input:* The input for this function will be attackable Animal cards.

*Output:* The output for this function will be sending attackable cards to the opponent player.

**Block:** This function occurs when the opponent tries to block the attacking card.

*Input:* The input for this function will be blockable cards for the player.

*Output:* The output for this function will be sending a blocked response to the player.

**Help:** This function occurs when the user requests for the help manual. The help function will load the manual in a new form.

*Input:* The input for this function will be the Game rules. Help manual will be available as a menu item in the Form.

*Output:* The output for this function will be game rules displayed on the new Form.

**Quit:** This function occurs when the users quit the application.

*Input:* The input for this function will be the decision of the player.

*Output:* The output will be quitting the current application and updating scores to the database.

**High Score List:** This function occurs when the user requests top five scores in the database.

*Input:* The input for this function will be database records.

*Output:* The output for this function will be top five score's records displayed on the screen.

### **3.8.2 Functional Requirements for the Conduit**

The conduit should be able to perform the following functions.

**Store Data into the Palm Database:** This function occurs when the Hotsync manager synchronizes the desktop data with the Palm database.

*Input:* The input for this function will be the data in the desktop file.

*Output:* The output for this application will be the desktop data converted into the Palm data format.

**Store data into the desktop file:** This function occurs when the hotsync manager synchronizes the Palm database with the desktop file.

*Input:* The input for this function will be the data in the Palm Database.

*Output:* The output for this function will be the data in the Palm database converted to the desktop format.

### **3.8.3 Functional Requirements for the Game Server**

The Game Server should handle the following functions.

**Create Deck:** This function occurs when the user requests for a new deck.

*Input:* The input for this function will be type of request, type of player and user name.

*Output:* The output for this function will be a new deck of thirty cards.

**Update Scores:** This function occurs when the user requests for updating his/her score.

*Input:* The input for this function will be new score, type of request, and user name.

*Output:* The output for this function will be the updated score list.

**Retrieve Top Five Scores:** This function occurs when the player requests the top five scores in the database.

*Input:* The input for this function will be the user's request.

*Output:* The output for this function will be the top five scores retrieved from the database.

### **3.8.4 Functional Requirements for the Database**

The Centralized database should perform the following functions.

**Store Records:** This function occurs when the server requests for storing of new records.

*Input:* The input for this function will be server request.

*Output:* The output for this function will be the storing of new records to the database.

**Update Records:** This function occurs when the server requests for updating existing records.

*Input:* The input for this function will be a server request.

*Output:* The output for this function will be updating records in the database.



# Chapter 4

## Design and Implementation

This chapter discusses the design and implementation phases of this project. The overview of the system architecture is explained first, followed by the design details of the individual components, and implementation details.

### 4.1 System Architecture

The main objective of this project is implementing a Pokemon-style card trading game, *Palm Maya*, in a distributed environment. Architectural Pattern allows us to design a distributed system using components that are independent of each other. We have to decide how to distribute the functionality among the components in order to optimize the usage of components and the resources involved. In this project, we have implemented a Three-tier Distribution Architectural Pattern [ADGKR] using Palm PDAs as clients, a Game Server, and a centralized Oracle database. This pattern will be discussed in the following sections.

#### 4.1.1 Three-tier Distribution Architectural Pattern

The *Three-tier Distribution Architectural pattern* is shown in the Figure 2. This pattern deals with partitioning application functionality into three tiers: front-end client, server, and Database. The server communicates with both the client and the central database.

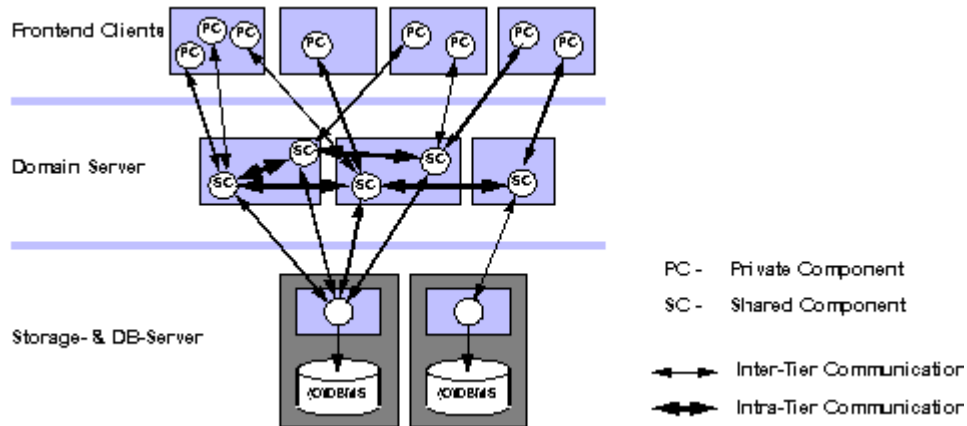


Figure 2: Three-tier Distribution Architectural Pattern  
Source: [ADGKR]

### 4.1.2 Use of Three-tier Distribution Architecture Pattern in our Project

The three-tier distributed system implemented in this project is shown in the Figure 3.

The core components of the distributed system shown in the figure are:

- Front-end clients tier: *“Palm Maya” Game, Conduit.*
- Server tier: *Game Server*
- Database-tier: *Central Database.*

The *Palm Maya* game is the actual game played on Palm devices. The main objective of the game is to always maintain the highest score among the players who played the game. To play the game, each player should have a deck of thirty cards stored in their Palm database. The *Game Server* will create a deck for each player. Once the decks are ready for the players, they will go for a toss - to decide who is going to start the game first - and start the game. The player can quit the game at any stage of the game. Once the player

quits the game, his score will be recorded in the *Central Database* through the *Game Server*.

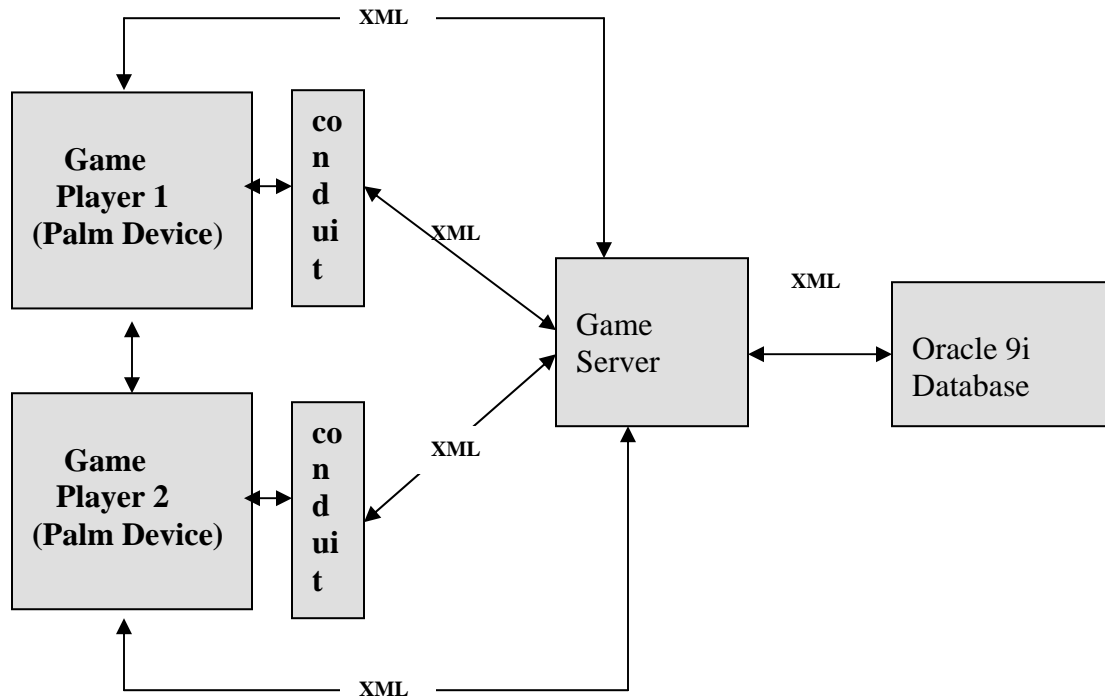


Figure 3: System Architecture

The *Game Server* is the essential logical component required in the distributed system. It accepts requests from the players and responds to their requests. The most important tasks of the *Game Server* are generating a deck for the current players, updating the score the current players, and retrieving the top five scores from the *Central Database*. It saves all the versions of the players' decks in the *Central Database*. The deck generated will be sent to the player through the *Conduit*. The deck is generated in XML file format. The *Central Database* is the Oracle database used for storing and retrieving XML data.

In our system, the *Game Server* resides on a desktop computer and the game is launched on a handheld computer running on Palm OS. The *Conduit* is used to exchange and synchronize data between these two devices during the hotsync operation. The *Conduit* will convert the data in the desktop computer to a Palm record format and store it in the Palm database. It will also convert the data in the Palm database to the desktop format and stores it in the desktop file.

## **4.2 Design**

This section of the report discusses the design of the distributed gaming system. The design of the individual components – the *Palm Maya* game, the Game Server, Conduit, and the database schema – are presented in the sections that follow.

### **4.2.1 Game Server**

The Game Server can handle requests from multiple players simultaneously. The main responsibility of the Game Server is handling all the data required for the game. The chief operations done by the game server are generating a new deck, updating the current player's score securely, and retrieving top five scores from the database. The operations handled by the Game Server are shown in the Figure 4. The individual operations are explained below.

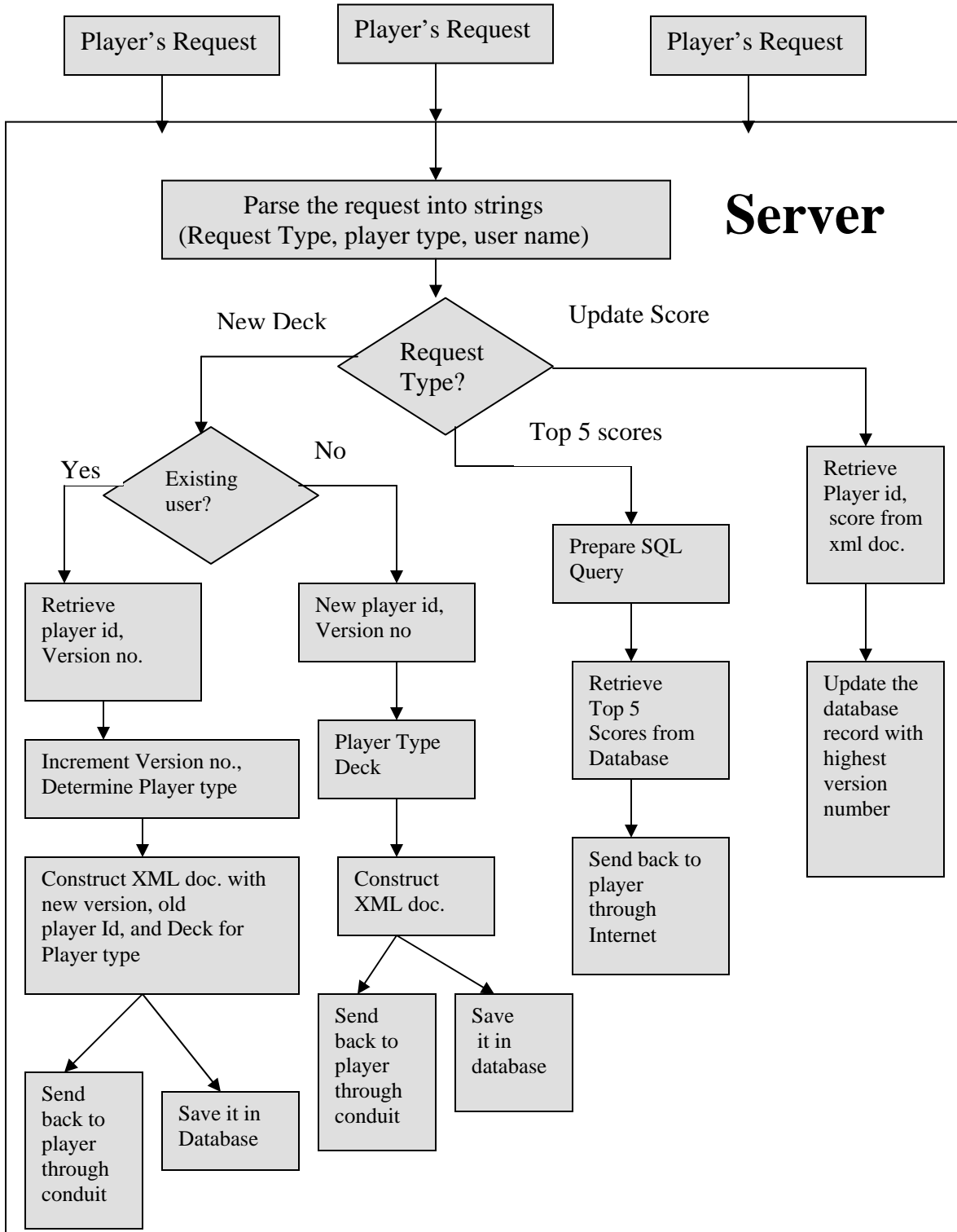


Figure 4: Game Server

#### 4.2.1.1 Create Deck

Each player should have his/her own deck of cards for playing the game. The deck should contain fourteen Land cards and sixteen Animal cards. The server will generate a deck containing both Land and Animal cards. The server receives the request from the user. The request includes request type, type of player, and user name. The user name should be an authorized user name. The server first checks for the request type. If the request type is *create deck*, the server will verify the existence of user in the central database.

*Existing user:* If the user already exists in the database, the server will retrieve the client ID and the deck version number from the database and increment the version number for that user. Depending on the player type, the server will generate a corresponding deck for the player. The deck will be constructed from the available cards in the text file. Each card in the text file is associated with a frequency ranging from 0 to 8500. The server generates a random number between 0 and 8500 to pick up a card associated with the frequency. This frequency is called required frequency. The server reads each card in the text file and adds up the frequency until it reaches the required frequency. Once the total frequency reaches the required frequency, the card with the required frequency is added to the deck. The same process will be repeated until thirty cards are added to the deck consisting of sixteen Animal cards and fourteen Animal cards. Then, the server will construct an XML document composed of client ID, version number, and the deck. The server will send the XML document back to the client through conduit and save it to the central database.

*New user:* If the user does not exist in the database, the server will generate a new client ID, version number, public key, and private key for that player. Depending on the player type, the server will generate the corresponding deck for the requested player. The same process for creating the deck for *Existing User* will be repeated here.

#### **4.2.1.2 Update Score**

If the players synchronize the game on the Palm device with the desktop through conduit, the score will be updated to the *Central Database* through the *Game Server*. The conduit writes the XML file containing the updated score and signature to the desktop. If the request type for the Game Server is *Update Score*, the Game Server will retrieve the client ID, score, and signature from the schema validated XML file using the XML Parser. The Game Server will check for the validity of the signature to ensure that the game played by the players is legitimate. Then the SQL query for updating the score will be constructed for a record with the given username and client ID. Each user has different versions of records in the Central Database. Always the player uses the latest version of the record. The score will be updated in the Central Database for the given player's latest record.

#### **4.2.1.3 Retrieve Top Five Scores**

Whenever the player wants to see the top five players' scores, he will request the server to retrieve the scores from the database. Each player has different versions of records in the database. Each version of the record is associated with a score the player achieved while playing that version of the deck. The server will send a SQL query to retrieve top

five scores from the database. The SQL query should be capable of doing following operations: sum up scores in all the versions of records for each player, sort the scores in descending order, retrieve top five scores in the sorted order. The retrieved scores will sent back to the player through Internet in XML file format.

## **4.2.2 “Palm Maya” Game**

*Palm Maya* is a multi-player game played on handheld devices running on Palm operating system. The main objective of the game is to always maintain the highest score among the players (who are playing the game). The basic requirement for starting the game is a deck of thirty cards stored in the Palm database. Along with the deck, the player’s details are also stored in the Palm database. The first record in the Palm database consists of player’s details – clientID, score, and version number - and the remaining thirty records are the Deck. Once the deck is available, the player will start the game by launching the game application on the Palm device.

The design of this game is further divided into two sections:

1. User Interface Design
2. Game Design

### **User Interface Design**

The main challenge in implementing the game on Palm devices arises due to small screen size. Form is the main user interface component used in the Palm OS applications. Form is the only way the user can interact with the application. Every Palm application must consist of at least one form. The Palm Maya application consists of six forms: Toss Form,



Initial Form, Server Form, Scores Form, Game Rules Form, and Reset Game Form. The Initial Form handles the entire user interface components required for the Palm Maya game. The Initial Form of the Palm Maya game application is shown in the Figure 5.

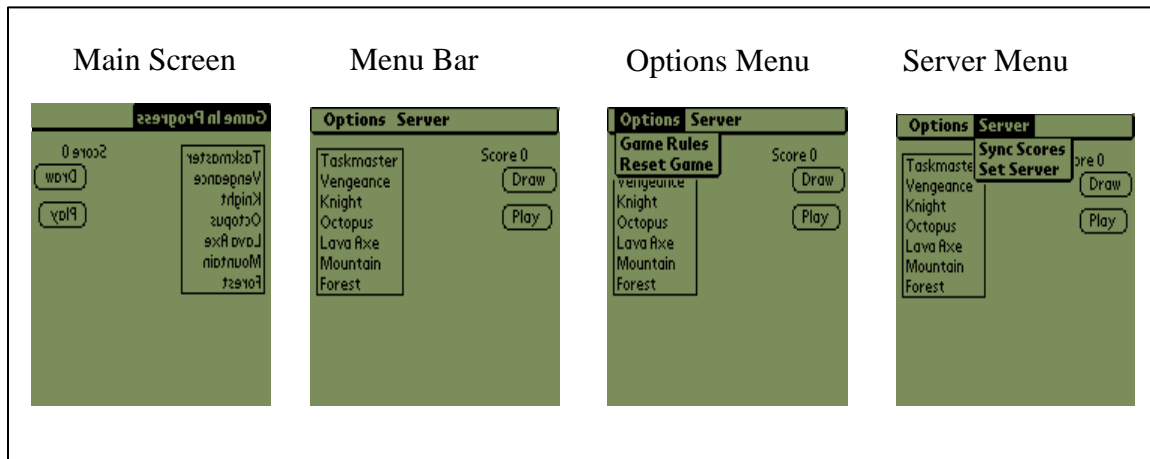


Figure 5: Initial Form

The Initial Form occupies the entire screen area of 160×160. The form contains three user interface components: List Interface, Buttons, and Menu Bar. The List Interface is used for displaying Handset of cards. Draw and Play buttons will do the operations corresponding to the Draw state and Play state in the game. Menu Bar displays two Menus: Options and Server. The Options menu has Game Rules and Reset Game menu items. The Game Rules menu item displays the Form containing the game rules. The Reset Game menu item displays the form that can handle both reset and quit operations. The Server menu has Sync Scores and Set Server menu items. The Sync Scores menu item displays the form containing top five players' scores. The Set Server menu item displays the form that can handle basic set up of the server.

## Game Design

The first step in playing the game will be to decide who is going to initiate the game. The player who wins the toss will start the game first. Each player's application will start with an initial set of seven cards known as Handset. The cards in the Handset are the first seven cards from the deck in the Palm database. The player will play the game using the cards in the Handset. The Handset may consist of both Land and Animal cards. If the player selects a card in the Handset, the player can see the type of card and characteristics of the card. There are four phases in the game: Draw, Play, Attack and Block, Update Score. To complete the game, the players' should go through all four phases in the game according to the rules specified in Chapter 4.

**Draw:** The draw state is the first state for every player playing the game. During each turn, the player must draw a card from the deck. The first seven cards from the deck are already drawn to the Handset. An index value is maintained to keep track of the cards in the deck. So, the card with current index from the deck is drawn and added to the Handset. Once the card is drawn from the deck, the index of the deck and Handset size are incremented to maintain the current status of the deck and Handset. The next state in the game will be decided based on the current state of the game.

If Land cards are available in the Handset, the player will be prompted for playing the Land card and go to the play state. If there is no Land card in the Handset and an Animal card is eligible for playing, the player will be prompted for playing the Animal card and go to the play state. If there are no Land cards and no eligible Animal cards in the

Handset for playing, and the Animal card is eligible for attacking the opponent, then the player will be prompted for attacking opponent and go to the combat state. If there are no Land cards, no eligible Animal cards, and no eligible attacking cards, the player's turn will be forwarded to the other player.

**Play:** During each turn, the player can play only one Land card from the Handset.

If the player's response is "yes" for playing a Land card, the player will play the Land card from the Handset and will add it to the Landlist. Then the Handset size is decremented and the Land list size is incremented to reflect the changes in the Handset. If the player's response is "no", the player's turn will be passed to the other player.

Land cards are used to pay for the Animal cards. If there are enough Land cards to pay for an Animal card, the player will be prompted for playing an Animal card. If the player's response is "yes" for playing an Animal card, the player plays the Animal card from the Handset and will add it to the Animal list. Then the Handset size is decremented and the Animal list size is incremented to reflect the changes in the Handset. The player can play any number of Animal cards, if he or she has enough number of Land cards to pay for them. For example, the player has five Land cards available in the Land list and he has two Animal cards with cost two and three respectively. In this case, the player can play both Animal cards in a single turn. If the players' response is "no" for playing an Animal card or the player does not have any attackable cards, the player's turn will be passed to the other player. If the player has attackable cards then the next state will be the Attack state.

**Attack and Block:** The player cannot attack with the Animal card that he had just played in the current turn. He can only attack with the Animal cards that he had played in the previous turn. If the player has attackable Animal cards, then he will be prompted for attacking an Animal card. The player can attack with a single Animal card or multiple Animal cards. If the player does not want to attack the opponent, then his turn will be passed to the other player.

**Single Card Attack:** If the player wants to attack the other player with a single Animal card, he will send his attacking card, along with the card's characteristics power and toughness, to the other player.

To block the attacking Animal card, the opponent should have blockable Animal cards. Any Animal card that is available in the opponent's Animal list is eligible for blocking the attacking Animal card. If the opponent has a blockable card, he will decide whether to block the attacking Animal or not. If the opponent is not willing to block the Animal card, then the turn will be sent back to the attacking player and his score will be incremented. If the opponent is willing to block the attacking card, then the trade will occur. The trading is based on the power and toughness of both the players. All possible cases for trading a card and results are shown in the Table 2. There are three possible results for trading a card:

1. Both players Animal cards will be dead
2. Attacking player's Animal card will be dead
3. Blocking players Animal card will be dead.

The notations used in Table 2 are explained below:

P1: Attacking Player's Power

T1: Attacking Player's Toughness

P2: Blocking Player's Power

T2: Blocking Player's Toughness

Table2: Trading Conditions

Trading Condition	Result
1. $(P1 > T2)$ and $(P2 > T1)$ 2. $(P1 == T2)$ and $(P2 == T1)$ 3. $(P1 < T2)$ and $(P2 < T1)$	Both Players cards will be dead
1. $(P1 > T2)$ and $(P2 <= T1)$ 2. $(P1 >= T2)$ and $(P2 < T1)$	Blocking Player's card will be dead
1. $(P1 <= T2)$ and $(P2 > T1)$ 2. $(P1 < T2)$ and $(P2 >= T1)$	Attacking Player's card will be dead

Source: Author

If *both the players are dead*, the blocking player's Animal card will be deleted from the Animal list first and then the control is transferred to the attacking player. The attacking player's Animal card will also be deleted from the Animal list. The attacking player's turn is over now, and the turn will be transferred to the opponent player.

If only the *blocking player is dead*, the blocking player's Animal card is deleted from the Animal list and a message is sent to the attacking player indicating that opponent's

Animal card is dead for this turn. The attacking player's turn is over now, and the turn will be transferred to the opponent player.

If only the *attacking player is dead*, a message is sent to the attacking player indicating that his or her Animal card is dead for this turn. The attacking player's Animal card will be deleted from the Animal list. Now, the attacking player's turn is over, and the turn will be transferred to the opponent player.

***Multi Card Attack:*** If the player wants to attack the other player with multiple Animal cards, he will send his attacking cards, along with their characteristics, power and toughness, to the other player.

The opponent can block one attacking Animal or all attacking animals based on his choice and availability of blockable cards. The opponent should always block multiple attacking cards with multiple blocking cards only. The opponent cannot block multiple attacking cards with single blocking card. If the opponent does not wish to block any of the attacking cards, then the attacking player will get highest score that is calculated based on the power of the attacking cards. The score here is sum of all cards power multiplied by ten. For example, there are two attacking cards and the opponent wishes to block only one card among the two attacking cards. Then the blocking card and attacking card will go for trading, and the other attacking Animal card that was not blocked will increase the attacking player's score. The trading process is similar to the process explained for *Single Card Attack* earlier.

**Beam and Receive:** A player communicates with the other player through Bluetooth or Infrared communication. The player can choose the way of communication he wants in the game. Each player sends his or her messages along with a signature to maintain the game security. The message and the signature are then verified using the public key provided. The player needs to communicate with the other player in the following situations:

1. To indicate that it is the receiving player's turn.
2. To send Animal cards and messages in the Attack and Block phase.

The player will send his/her opponent a record containing the mode, the Animal card's name, power and toughness. Depending on the receiving mode, the opponent's action will be decided.

There are five modes in this game:

- N - To indicate that it is the receiving player's turn
- A - To indicate that other player is attacking
- CB - To indicate that opponent cannot block the attacking Animal
- OD - To indicate that opponent is dead
- D - To indicate that Attacking player is dead.

Once the player has decided the way of communication, he can send a single record or multiple records depending on the situation. In case of attacking with multiple cards only, the player will send multiple records.

**Update Score:** If trading does not occur during the Attack and Block phase, the attacking player will get the score according to attacking player's Animal card power. The score is

calculated as the power multiplied by ten. The score from the current turn will be added to the total score for the game.

**Help Menu:** The players should play the game according to the rules provided in the help menu. The Game rules are described in Section 3.1.

**Quit:** The player can quit the game at any stage of the game. When the player quits the game, the player's record is retrieved from the Palm database. The score in the database record is modified with the current score and saved back to the database. Once the record is stored in the Palm database, the game application will close. When the Hotsync manager synchronizes the game application with the desktop, the Game Server will update the score in the Centralized database.

**Top Five Scores:** Whenever the player wants to view the top five scores in the Centralized database, he or she will request the Game Server for the top five scores. The Game Server will retrieve the top five scores from the database and then sends it back to the player in XML data format through Internet. The data sent by the server is parsed and displayed on the Palm's form. The connection between the Game Server and the Palm device is established through Palm Networking and Internet.

### **4.2.3 Game Conduit**

Game Conduit is the Palm conduit implemented to synchronize data between desktop and Palm database. The important operations done by the Game Conduit are synchronizing



the data on the desktop file with and Palm database and vice-versa. The operations done by the Game Conduit are explained in the following sections.

#### **4.2.3.1 Store Data into the Palm database**

The Game Server generates a deck for each player and stores it on the desktop in the XML file format. The XML file contains player's details like client ID, score, and version number; and the deck of cards. Each deck element in the XML file has four properties: name of the card, power of the card, toughness of the card, and a bitmap image representing the card. Each record in the XML file also has four properties indicating whether it is new, updated, archived, or deleted record. Only the record with new tag will be stored into the Palm database. The conduit will read each element in the XML file and store the data into temporary records. The bitmap image is also converted from Hex to Bytes and stored in the temporary record. All the data stored in the temporary records is converted into the Palm record format. Records with new tag value of true are transferred to the Palm database when the Hotsync manager synchronizes the application.

#### **4.2.3.2 Store Data into the Desktop file**

Whenever the player quits the game, his or her score will be updated in the Palm database. The modified score in the Palm database is updated into the Centralized database through the Game Server when the Hotsync manager synchronizes the application with the desktop. Each record in the Palm database will have four tags indicating whether it is new, updated, deleted, or archived.

The Game Conduit will read the raw data in the Palm database and stores it into temporary records. The dirty bit for the record in the Palm database is set to true to represent the modified data in the Palm database. The data in the temporary record with modified or new tag value of true is copied into the desktop file, when the Hotsync Manager synchronizes the application.

## **4.3 Implementation**

This section of the report discusses the implementation details of the project. The main components of the Distributed Gaming system are the following:

1. Game implemented on Palm device,
2. Game Conduit,
3. Game Server, and
4. Centralized Oracle database.

Implementation of the Distributed Gaming System and Design Patterns used in implementing the project are explained in following subsections.

### **4.3.1 Design Patterns Used**

We have implemented object-oriented design patterns, game design patterns, and usability patterns to maintain the quality of the project. The object-oriented pattern, Observer pattern, is used for implementing the Game Server. Game design patterns - State-based pattern and Trading pattern - are used in implementing the game. The usability pattern, Multi-channeling, is used in providing communication between Game Server and Palm device.

### **4.3.1.1 Observer Pattern**

This pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. [ERRJ02]

This pattern is implemented in this project in the Game Server. We used delegation in place of inheritance and regained our ability to extend another class. This pattern allows low coupling between *GameServerThread* and the *GameServer* class. The *GameServer* class *start()* method instantiates the *GameServerThread*. This class extends the Thread class and its run method listens to all incoming connections.

### **4.3.1.2 State-based Pattern**

The state-based pattern is similar to object-oriented state pattern. This pattern is useful in implementing applications that can handle different logically dependent states. For example, the different states handled by a Graphics editor program are Move, copy, and draw shape.

In this project, this pattern is implemented in the Palm Maya Game. The different states in a game are implemented using state-based pattern. The state-based pattern determines the current state of the game and the next states (moves) in the game depending on the current state. Implementing this pattern would be helpful to the players because the players would be prompted for playing the next state.

### **4.3.1.3 Trading Pattern**

Interactive games need a mechanism to simulate social interaction between players. To trade with each other, players need to communicate with each other. This type of pattern can be seen in Pokemon-style and other card games.

This pattern is implemented in this project in the “Attack Phase” of the Palm Maya Game. The Trading Pattern implements the rules of the game for attacking/blocking the opponent. When the player decides to attack, the player sends a message to the opponent. If the opponent has eligible blocking cards and decides to block, then trading activity begins.

### **4.3.1.4 Multi-channeling**

It would be helpful to provide users a mechanism to access a system using different types of input/output devices. For example, one can access auction sites like eBay.com from a desktop computer, a mobile phone, a PDA, or using interactive TV. This pattern is implemented in this project in the Game Conduit and the Game modules to provide communication between the Game Server and the Palm device. The Game Conduit provides communication through the Palm Cradle and the Game Module provides the same through a mobile phone (and Palm OS network library). The Game Conduit is used to get the deck generated by the Game Server to the PDA and to update the player’s score in the central database. The Game module uses this pattern while retrieving the top five scores from the centralized database.

This type of pattern is implemented in the project because it would be very helpful for the user to have multiple data access mechanisms. For example, the user can efficiently transfer large amounts of data through the Palm Cradle, during the initial setup of the game. Later, the user can use a mobile phone to dynamically access small amounts of data, like the top five scores, while playing.

### **4.3.2 Game Server**

The *Game Server* is implemented using the *Observer Pattern*. The Game Server needs to handle requests from multiple clients simultaneously. The main responsibilities of the Game Server are generating a deck of cards, updating and retrieving scores for players. The *Game Server* is implemented using Java Network Programming. The powerful network and multi-tasking (threads) capabilities of java make it an ideal language for developing servers that can handle multiple clients simultaneously while providing platform independence. Java Database Connectivity is used for communication between the *Game Server* and the *centralized Oracle database*. The Class diagram for Game Server is shown in the Figure 6.

The *GameServer* listens for new clients, gets their socket objects, and passes the information to separate threads; in our case, it is called *GameServerThread*. The *GameServerThread* class extends the *Thread* class and implements the runnable interface. Each *GameServerThread* will handle a separate client; its *run()* method creates a loop for listening to messages from the client. All the logic of the server is implemented in the *GameServerThread* class. The core of the *GameServerThread* class is its *run ()*

method, which invokes a loop for getting input from the client. The input from the client is handled by the *handleNewInput()* function.

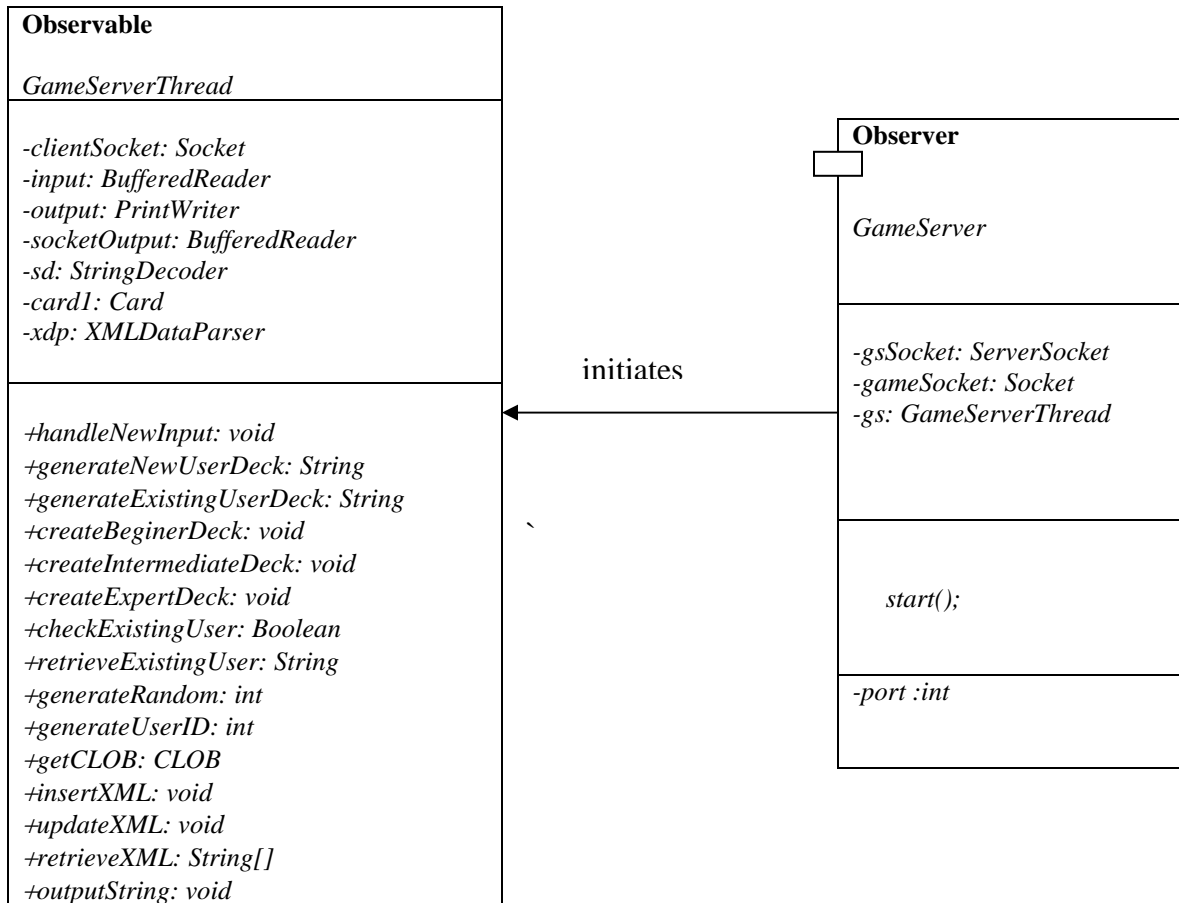


Figure 6: Class diagram for Server

The client can send three different types of requests: NewDeck, UpdateScores, GetScores. Along with the request type, the client will also send authorized username, and Player type. The *StringDecoder* class parses the input string from the client and finds the request type. Based on the request type, the *handleNewInput()* function will delegate the responsibility to other functions.

## **New Deck**

If the request type is *NewDeck*, the function *checkExistingUser ()* is called to check whether the user is a new user or an existing user. If the user is a *new user*, the *generateNewUserDeck()* function is called to generate a new record for the user. The user's record consists of clientID, username, score, version number, and deck of thirty cards. The clientID of the user is generated randomly using *generateUserID ()* function. The version number starts with number one and is incremented for each version of the record for the same user. The deck of thirty cards is created based on the Player type. If the player type is *Beginner*, the *createBeginerDeck ()* function creates the beginner deck. If the player type is *Intermediate*, the *createIntermediateDeck ()* function creates the intermediate deck. Finally, if the player type is *Expert*, the *createExpertDeck ()* function creates the Expert deck for the player.

If the user is an existing user, the clientID and version number of the record are retrieved from the database using *retrieveExistingUser ()* function. The *generateExistingUserDeck ()* function is called to generate a new version of the record for the existing user. The deck is generated for each player depending on the player type.

Whether the player is an existing or a new user, the record is generated in the XML file format. The XML Parser is used to generate the record in the XML file format. Once the deck is generated for the players, the deck needs to be sent back to the player and also needs to be stored it in the database. The *outputString ()* function handles the sending of

the data back to the client. The *insertXML ()* function handles the storing of XML data into the database.

The generated XML file contains more than 4000 characters. In order to store this large amount of data into the centralized database, the *getCLOB ()* function creates a CLOB object and stores it into the database. The XML document is validated against the XML schema, both in the database and at the client.

### **Update Scores**

If the request type is *UpdateScore*, the *XMLDataParser* class is instantiated. The *XMLDataParser* class validates the player's XML document against its schema, parses the XML document, and retrieves the player's score, clientID, message, and signature from the XML document.

The *updateXML()* function handles updating of the score in the centralized database securely. The *updateXML()* function also handles verification of messages and signatures using private key provided in the database. Then the SQL statement to update XML data in the database is constructed in the *updateXML()* function to store player's score in the centralized database. The XML feature XPath is used to traverse the XML document in the database.

### **Retrieve Scores**

If the request type is *RetrieveScores*, the *retrieveXML()* function is called to retrieve the top five scores from the database. The SQL statement to retrieve the data from the XML



document is constructed in the *retrieveXML()* function. The XQuery and XPath features of XML are used to retrieve the data from the database.

### **4.3.3 Palm Maya Game**

The Palm Maya game is implemented as a State-based pattern to represent moves from one state to other state. Trading Pattern is implemented in the Attacking and Blocking phase. The functionality of the game and its rules are described in Chapter 3. The implementation details are explained in the following sections.

#### **4.3.3.1 Initial Setup**

The Hotsync Manager should be used to install the application on the Palm device. After installing the application, during the next hotsync, the Game Conduit synchronizes the deck of thirty cards with the Game application installed on the handheld device. After the player launches the application, the first form opened is the *TossForm*. *TossFormHandleEvent ()* function handles all events occurred in the *TossForm*. Once the toss is decided, the player who won the toss has to play first. The player who won the toss will be prompted to begin the game. The player who loses the toss will wait for the other players to play first. Then the player's control will be transferred to the *MainForm*. The *TossForm* and the *MainForms* of both players are displayed in Figure 7.

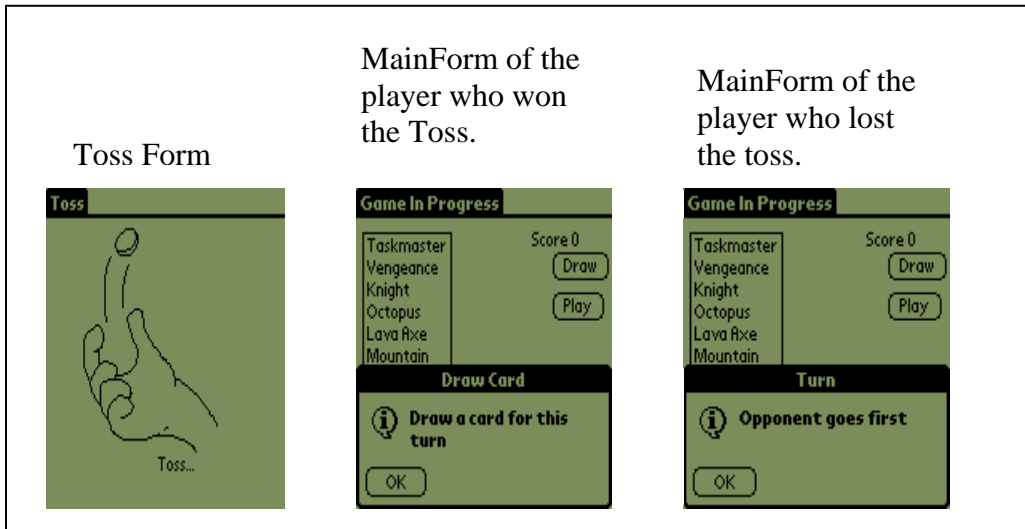


Figure 7: TossForm and MainForms in the Game

*MainForms* of both the players are opened with an initial set of seven cards from the database. Handset is the initial set of seven cards from the database. *CopyDatabaseRecord ()* function handles the copying of database records to the *currentList*.

#### 4.3.3.2 State-based Pattern

The game is implemented as a State-based pattern. The state diagram for the game is shown in the Figure 8.

##### Draw State

The player who wins the toss will start with the *Draw* state. To draw a card from the deck in the database, the player should tap on the *Draw* button. If the player taps and releases the button on the *MainForm*, a *ctlSelectEvent* enters the event queue and handles all the button events. All button events are handled in *ButtonHandleEvent ()* function. The *DrawButton* case in the *ButtonHandleEvent ()* function handles the *Draw* state in the game.

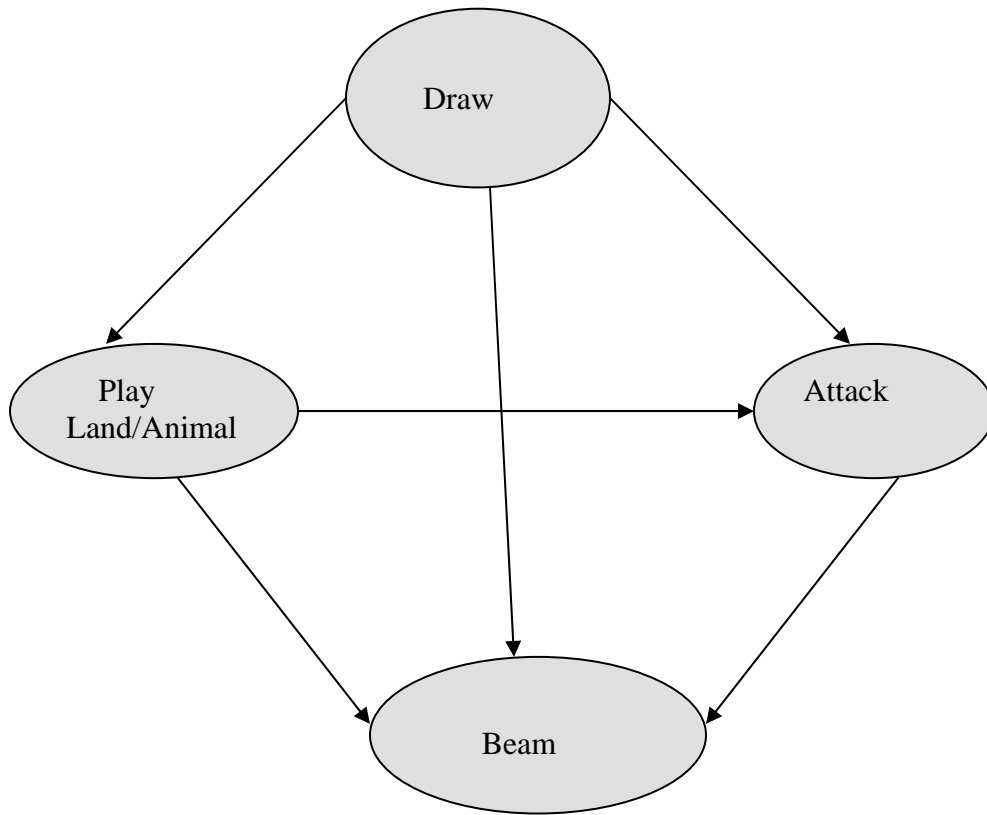


Figure 8: State diagram representing the game

The *index* is maintained to keep track of deck in the database. During the Draw state, the card with the index value is retrieved from the database and is added to the Handset in the ListInterface. If the deck in the database is completely exhausted, the *index* value points to the first card in the Deck. *Draw* state is the most crucial state in the game as it can determine the next state in the game. The operations handled by the *DrawButton* case are explained below:

1. *Ensure it is the current player's turn:* If the current turn of the game is not the player's turn, then the player will not be allowed to play the game further. The player can view the characteristics of his/her card in the *ListInterface* by selecting the cards in the Handset.

2. *Make sure the card is not already drawn:* If the player already draws the card from the deck, he will not be allowed to draw more cards from the deck.

3. *Draw a card from the deck and add it to the List*

4. *Check for availability of Land cards in the deck:* If there are any *Land cards* available in the Handset, the player will be prompted for playing the land card and the game will go to the *Play* state.

5. *Check for the availability of Animal cards, if there are no Land cards:* If there are no Land cards in the Handset, and there are eligible Animal cards; the player will be prompted for playing an Animal card and go to the *Play* state. *MoreAnimalCardsPlayable ()* function will check for the availability of eligible Animal cards. If eligible Animal cards are not available, it will check for eligible attacking cards. If there are attacking cards available, the game will go to the *Attack* state.

6. *No Land cards, No eligible Animal cards, No attackable cards, go to Beam state:* If there are no cards available for this turn, the game will go to the Beam state indicating that the player's turn is over. The *penLock* and *buttonLock* variables are set to *true* before sending the Beam. This indicates that the player cannot make any move once his/her turn is over.

7. *Redraw the MainForm with updated capabilities:* The player cannot attack with the Animal card that he/she just played in the current turn. So, the Animal cards played in the previous turn are made eligible for attack in the current turn. If Animal cards were played in the previous turn, then only the *animalSize* would be greater than zero.

## **Play State**

During the *Play* state, the player can play either a Land card or an Animal card. If the player does not have Land cards and eligible Animal cards, he/she will go to the *Beam* state and his/her turn will be over. If the player has attackable cards, he/she will go to the *Attack* state. The operations handled by the *PlayButton* case are explained below:

1. *Play Land Cards*: The player must have a Land card to play this turn. First, a Land card is selected from the Handset using *LstGetSelection ()* function. The *SelectedIndexIsLandCard ()* function then makes sure that the selected card is a Land card. If the selected card is a Land card, it is copied from the Handset to the *LandList* using *CopyToLandList()* function. Once the Land card is copied to the *LandList*, *landSize* is incremented and Handset size is decremented to reflect the changes. The List is updated using *SetList()* function. The player can play only one Land card during each turn.

2. *Draw (Display) Land Cards*: The land cards played are drawn on the screen using *reDrawLandList()* function. The function draws a bitmap representation of the Land card on the screen. Land cards are not placed side by side to save the screen area. Instead, the count of Land cards is displayed on the corner of the bitmap image.

3. *Play Animal Cards*: Once the player played the Land card, his Land count is incremented (including this turn and the previous turn). The *isAnimalCardPlayable ()* function checks for the eligibility to play an Animal card. If there are enough land cards to pay for the Animal card, the Animal card can be selected from the Handset using *LstGetSelection()* function. The *SelectedIndexIsAnimalCard()* function is used to check whether the selected card in the Handset is an Animal card or not. The

*CopyToAnimalList()* function is used to copy the Animal card in the Handset to the Animal list. Once the card is added to the Animal list, the Animal size is incremented and the Handset size is decremented to reflect the changes. The *SetList()* function is used to update the current List. The player can play more than one Animal card during the turn if he/she has enough number of Land cards to pay for the multiple Animal cards. The *usedLandCount* is maintained to keep track of how many land cards are remaining.

4. *Draw (Display) Animal Cards:* Animal cards are displayed side by side on the screen. The *DrawImageAL()* function is used to draw a Animal card image on the screen. The system will not accept more than three Animal cards on the screen. If there are attackable cards, a small dotted rectangle is drawn around the bitmap image to represent it is an attackable card.

Example: The state of the Game at the end of the Play state is shown in the Figure 9. The Player has two Land cards. He/she played the Animal card with cost two. Figure 10 represents the state of the player during the beginning of the next turn (Draw state). In Figure 9, the Animal card knight is just drawn. The small rectangular frame around the Knight in Figure 10 represents that Knight was played in the previous turn and is eligible for attack during this turn.



Figure 9: Previous Play state



Figure 10: Current Draw state

## **Attack State**

During the *Attack* state, the interaction with the other player should occur. Once the player decided to attack the opponent, the game will go to the Beam state and player's decision is sent to the opponent. If the opponent decided to block the player, trading will occur between the players. Different cases of trading are implemented using *Trading Pattern*. Trading depends on the Power and toughness of both players' animals. The actions of the attacking and blocking players are discussed in the following sections.

### ***Attacking Player:***

If the player decided to attack the opponent, he/she will select the attackable card by tapping it. Then the *penDown* event queues the Event queue. If there are multiple attackable cards, the player will be prompted for attacking with all available cards. The operations handled by the attacking player are described below:

*1. Handling penDownEvent and Beam attacking cards:* If the player taps the screen in the attackable card area, the *penDownEvent* will occur. The screen area is recognized using the *RctPtInRectangle ()* function. The program will handle attacking with any number of cards. The *MoreAnimalCardsAttackable ()* function will handle attacking with multiple cards. All the attacking cards are added to the array and beamed to opponent using *BeamRecord ()* function. The attacking cards are marked with letter 'A' on the screen to indicate that the card is attacking using *WinInvertChars ()* function.

*2. Update the score:* If the opponent does not have blockable Animal cards, the opponent will beam the player with mode 'CB' indicating that the opponent cannot block the attacking Animal. Then the player's score will be incremented. The *HandleAttack()* function increments the score.

3. *Update the Animal List*: If the opponent blocks the Animal card and the player's Animal dies during the trading, the opponent will beam the player with mode 'D' indicating that the player's Animal card is dead during this turn. The Player's Animal list is updated using *UpdateAnimalList ()* function.

Example: The attacking player's actions are shown in the Figure 11. The player has two cards, Knight and Hammer. Knight is an attackable card and Hammer is not an attackable card. First, the player will be prompted for attacking an Animal card. If the player's response is 'Yes' and he/she taps the Knight card, the attacking card, Knight, will be sent to the opponent. The Knight card is marked as 'A' indicating that it is attacking.

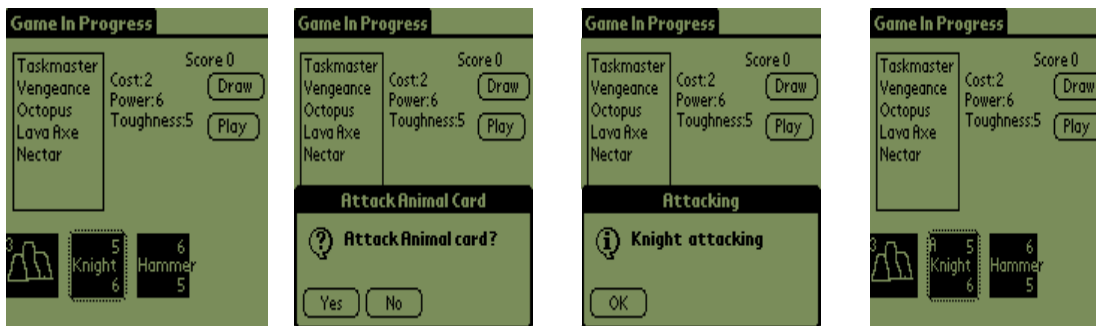


Figure11: Attacking state

### Blocking Player:

Once the opponent receives the records from the player, the mode will be checked by *retrieveRecord()* function. If the mode is 'A', the opponent has to respond to the attack. The operations handled by the blocking player are explained below:

1. *Check for Blockable cards*: To block the attacking card, the opponent should have blockable cards. The *MoreBlockableCards ()* function will check for the availability of blockable cards. Suppose there are three attacking cards, the opponent might have three



or less than three blockable cards. If he has three blockable cards, depending on his strategy, he will block each attacking Animal card. Then the control will be transferred to the *HandleAttack()* function to handle the attack.

2. *Check for trading conditions and handle the attack:* Once the opponent decided to block the attacking player, the result of the trading will depend on the power and toughness of the Animal cards of both the players. The trading conditions implemented are explained in Table 2. The *HandleAttack()* function handles all possible cases that can occur during trading between players. The three possible results that can occur during trading are following:

- Animal cards of both the players are dead
- Attacking player's Animal cards are dead
- Opponent's Animal cards are dead.

3. *Send the result of trading back to the Player:* Whether the attacking player loses or wins the trade, the result of trading will send back to the attacking player.

Example: The blocking player's actions are shown in Figure 12. When the player attacks with Knight, the opponent blocks with Nector, which has more power and toughness. Then the attacking player's card is dead and removed from the Animal list displayed on the screen. Only Hammer is displayed on the Attacking player's screen.

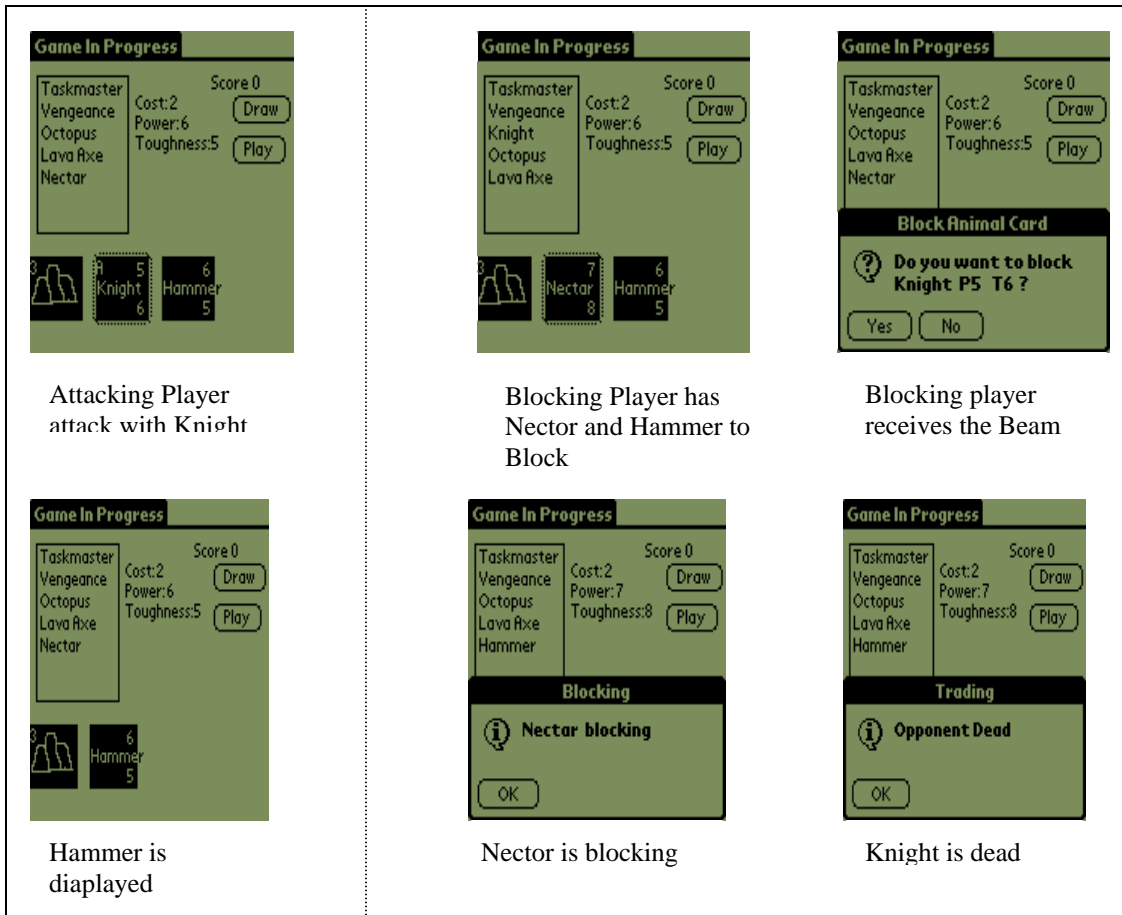


Figure 12: Trading between Players

## Beam State

Players communicate with each other either through Infrared or Bluetooth communication. The player can choose the method of communication, and that will be used for throughout the game. Both Infrared and Bluetooth communication use the Palm OS Exchange Manager functions.

We differentiate the way of communication using Exchange Library URL schemes. The URL *\_btobex://?\_single/* is used for Bluetooth communication, and the URL *\_beam:* is used for Infrared communication.

The data exchanged between players is in the form of a record format. During the Draw state and Play state, single records are sent between the players. Multiple records are sent between the players only when they are in Attack state and attacking with multiple cards.

The operations handled by the Beam state are explained below:

*1. Beaming Single Record and Multiple Records:* Beaming a single record or multiple records is handled in the *BeamRecord()* function. The *ExgSocketType* structure variables are initialized to specify the Creator ID of the receiving application, and to specify the type of data transfer between devices. The *SendBeamedData ()* function is called to transfer the data between devices. The first transfer should be the number of records to be transferred between the players. Once the number of records being beamed is transferred, one can send the data for the actual record. The Palm Exchange Manager function *Exgput()* establishes the connection between the devices. The Palm Exchange Manager function *ExgSend()* transfers the actual data between the devices. Once the data transfer is completed, *ExgDisconnect()* function disconnects the connection between the devices.

*2. Receive Single Record or Multiple Records:* Receiving a single record or multiple records is handled in the *ReceivedRecord ()* function. The Exchange Manager sends a *sysAppLaunchCmdExgReceiveData* launch code to the receiving application. The *ReceiveRecord()* function is called in the *PilotMain()* function corresponding to the launch code. The total number of records is received first and then the actual data corresponding to the records is received. The *ImportData()* function is called to receive the data sent by the other device. The Palm Exchange Manager function *ExgAccept()* accepts the incoming data. The *ExgReceive()* function receives the actual data into

temporary records. Once the data is received, *ExgDisconnect()* function disconnects the connection between the devices.

3. *Generating Digital Signature*: A trivial digital signature scheme is implemented to maintain the game security. The *createSignature()* function handles the generation of digital signature for the messages transferred between the players.

#### **4.3.3.3 Top Five Scores**

Whenever the player wants to view the scores of the top five players in the centralized database, he will request the Game Server to retrieve the scores from the database. The connection between the Palm device and the Game Server is established through *TCP/IP* and *Palm OS Net Library*. The data from the Game Server should be accessed through the Internet. The request from the Palm device and the response from the Game Server are handled in *DownloadScores()* function. The *DownLoadScores()* function calls the *InitNetwork()* function to set up the connection with the Internet service provider. Once the network connection is initialized, the hostname of the computer running the server is resolved into a 32-bit *IP address* in the *GetIPAddress()* function. The *NetLibSocketConnect()* function makes a connection with the Game Server. The actual request to the Game Server is handled by the *SendData()* function. The *SendData()* function sends the 'GET' request to the Game Server using *NetLibSend()* function. The *ReadLine()* function handles receiving top five scores from the Game Server. The Palm OS Net Library function *NetLibReceive()* is used to receive the actual data from the server. Then the player can see the scores of the top five players in the centralized database by selecting the *Top Five* Menu item on the *MainForm*.

### 4.3.3.4 Application Preferences

The Palm OS supports only the execution of one application at a time. Once the game application is launched and running, if the player accidentally opens another application and returns to the game application, the game will start from the initial state. The player loses the current state in the game. To avoid this state, application preferences are used in this application. The Application preference is a kind of small database. The *PrefSetAppPreferences ()* function is called in the *StopApplication ()* function to save the state of the game in the Application Preferences before closing the application. The *PrefGetAppPreferences ()* function is called in the *StartApplication()* function to retrieve the state of the game while re-opening the application.

Example: If the player is playing the game and he is in the *Play* state and suddenly closes the application, the state of the game is saved to the preferences database. When the player re-opens the application, the application will open at *Play* state. Figure 14 represents the example.

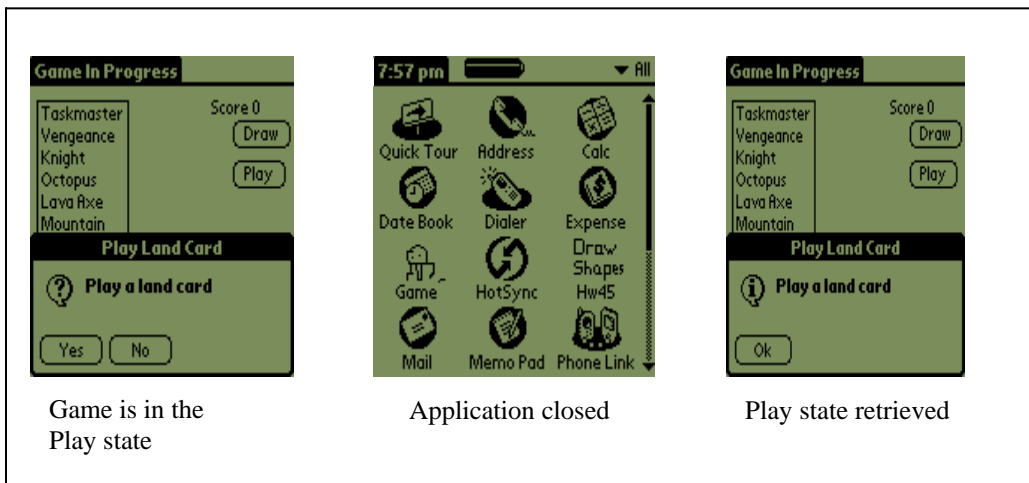


Figure14: Application Preference

### 4.3.3.5 Quit Application

The Player can quit the Game application at any state of the game. To quit the game, the Player has to choose *Reset Game* Menu item. Then the *ResetForm* will be opened. By tapping the *Quit* button on the *ResetForm*, the player can quit the application. When the player taps on the *Quit* button, the player's score so far is updated to the Palm database and the application will be closed. The *SaveScore ()* function handles database operations required for updating the player's score. After the player's scores are updated to the database, the *appStopEvent* enters the event queue and the application will be stopped.

Figure 15 shows quitting the game application.

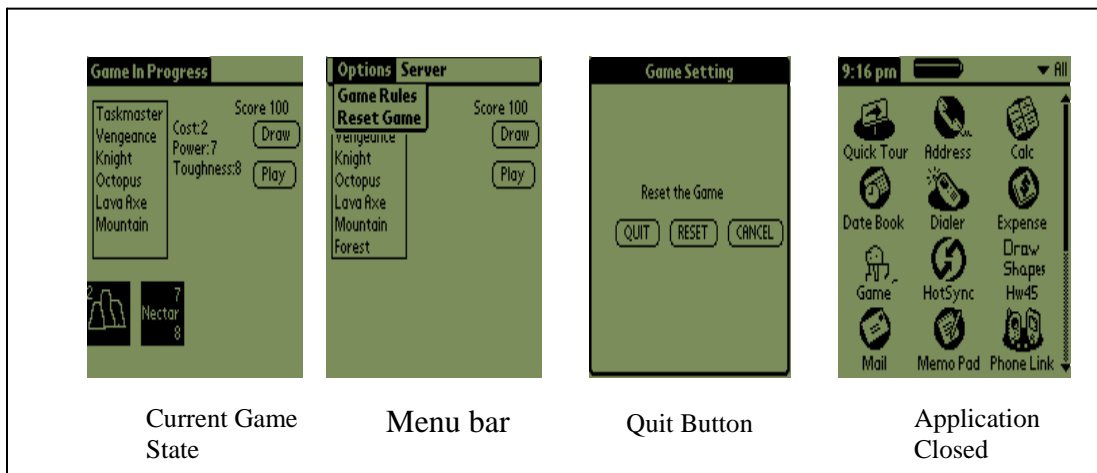


Figure 15: Quit the Game

### 4.3.3.6 Game Help

Players playing the game for the first time need to understand game rules. To view the game rules, the player need to select the *Help* Menu item on the *MainForm*. Then the *HelpForm* will be opened with rules. Figure 16 shows the *HelpForm* for the game application.



Figure 16: Game Rules

### 4.3.4 Game Conduit

The Game Conduit handles the actual transfer of data between a handheld application and a desktop data source during a Hotsync operation. The various operations handled by the conduit are explained in this section.

#### 4.3.4.1 Synchronize Desktop data with Handheld Application

When the player requests the Hotsync operation, the first function to be called is *RetrieveDB()*. The *RetriveDB()* function reads the record and adds it to the Handheld database. The *RetrieveDB()* function transfers all the functionality to the *ReadRecord()* function. The *ReadRecord()* function reads the data from the desktop file and stores it into temporary structure variables representing the Handheld record. In addition to the data, the Deck record needs to convert bitmap in Hex decimal format to bytes. The *ReadRecord()* function will call the *ConvertPctoGeneric()* function to convert data stored in the temporary structure variables to a Palm record format. Both structures representing the player's details (the first record) and the Deck elements (the remaining 30 records) are defined in Table 3:

Table 3: Structs for Deck and Player's Details

Player's Details struct	Deck Element struct
<pre> struct KeyAppInfoType {     DWORD recordID;     bool newRec;     bool updated;     bool deleted;     bool archived;     char clientID[8];     char randInit[2];     char score[4];     char version[4]; }; </pre>	<pre> struct CardsPCRecord {     DWORD recordID;     bool newRec;     bool updated;     bool deleted;     bool archived;     char cname[20];     char cost[4];     char power[4];     char toughness[4];     char typeOfCard[4];     BYTE bitmap[32][4]; }; </pre>

Source: Author

Once the data is available in the temporary structure variables defined in Table 3, it will be converted to Palm record format by the *ConvertPCtoGeneric()* function. Memory is allocated depending on the size of record and each struct variable is converted to the Palm record format and stored in the memory location. The basic operations of the function *ConvertPCtoGeneric()* are *memset()*, *memmove()*, and *setRawData()*.

#### 4.3.4.2 Synchronize Handheld data with Desktop data source

When the player requests the Hotsync operation, the first function to be called is *StoreDB()*. The *StoreDB()* function stores the data into the desktop file in XML format. The *StoreDB()* function calls the *WriteXMLHeader()* function to write the header for the XML document. Once the header is written, the records to be stored are the player's details record and the deck of thirty cards. The *StoreDB()* function transfers all control to the *WriteRecord()* function.



The *WriteRecord()* function handles the conversion from Palm record format to desktop format by calling *ConvertGenericToPC()* function. Once the *ConvertGenericToPC()* function converts the Palm data format to the desktop data format, the converted data will be written back to the desktop using the *WriteRecord()* function. The bitmap data is also converted to HEX format by the *WriteRecord()* function. Once all the records are written to the desktop file, the function *WriteXMLEndDocument(void)* is called to write the XML end statement.

### 4.3.5 Centralized Database

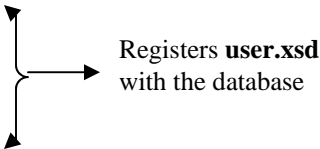
The operations handled by the centralized Oracle database are explained in this section.

#### 4.3.5.1 Store Records

The Game Server generates a record for each player in XML format and stores it into the centralized Oracle database. The XMLType table XML\_USER is created to support XML records. The XML schema validating the XML records is registered with the database first. Whenever the new record is stored into the database, it is validated against the schema. If the record is valid, then only it is allowed to be stored. The player in the database is identified by the combination of username and clientID. The SQL statements used for registering XML schema, creating table, and storing data are explained below:

##### 1. Register XML Schema

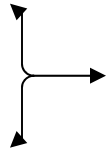
```
begin
  dbms_xmlschema.registerSchema
  ('user.xsd',
   getDocument('user.xsd'),
   TRUE, TRUE, FALSE, FALSE
  );
end;
```



Registers **user.xsd** with the database

## 2. Create XMLType Table

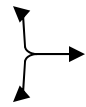
```
CREATE TABLE XML_USER of XMLType  
XMLSCHEMA "user.xsd"  
ELEMENT "user";
```



Creates **xml\_user** table  
which can access only  
schema validated XML  
documents

## 3. Store XML document into XMLType table

```
INSERT INTO XML_USER  
values (xmltype(getDocument('user.xml')));
```




Inserts **user.xml**  
document into  
**XML\_USER** table

## 4.3.5.2 Update Records

The Game Server updates the latest scores of the players into the centralized Oracle database. Oracle database allows XPath and Xquery features of XML to traverse XML documents in the database. To update the score in the XML document in the database, the query traverses the XML document using XPath up to the score node and updates the value for that node. The update statement used for updating the score in player's record in the database is shown below:

### 1. Update Score

```
UPDATE XML_USER x SET value (x) =  
updateXML (value (x),'/user/score/text ()','updatedScore')  
WHERE existsNode (value (x),'/user [clientID="clientID"])=1  
AND extractValue (value (x),'/user/version')  
in (select MAX (extractValue(value(x),'/user/version'))  
from xml_user x  
WHERE existsNode (value (x),'/user [clientID="clientID"])=1);
```



Updates given  
player's score

## 4.3.5.3 Retrieve Records

When the player requests the top five scores from the central database, the Game Server retrieves them from the database and sends them back to the player. User names and their

respective scores are retrieved from the database for the top five players. The SQL statement used for retrieving top five scores from the database is shown below:

*1. Retrieve top five scores*

```
SELECT * FROM  
(SELECT extractValue (value (x),'/user/userName') "MyUser",  
SUM (extractValue (value (x),'/user/score')) "Score"  
FROM xml_user x  
GROUP BY extractValue (value (x),'/user/userName')  
ORDER BY SUM (extractValue (value (x),'/user/score')) DESC)  
WHERE rownum < 6;
```



Retrieves Top  
five scores from  
database

## Chapter 5

### Usability Testing

Usability testing is a way of measuring how well users can actually use the product for its intended purpose. Game usability is different from other applications. Fun is the main factor that differentiates the game usability from usability in other frame works. Usability provides the framework for playability to maintain the quality of the game. Playability measures the quality of the game based on the interaction style, fun, and game navigation. The main purpose of testing the game playability is to test the game from the user's perspective. Playability is affected by the quality of the story line, responsiveness, usability, pace, customizability, control, intensity of interaction, intricacy, and strategy, as well as the degree of realism and the quality of the graphics and sound.

To test the distributed gaming system developed in this project, we have selected three groups of users with various backgrounds. All the users have expertise in playing games on Palm devices. The players were given a brief overview of the game before the testing commenced. The following are the important criteria in observing the user:

1. Can users easily understand the game setup?
2. Can users easily accomplish the intended tasks at their intended speed?
3. Do users make any mistakes when they interact with game interface? Can the users recover easily from their own mistakes?

4. Can users understand the communication mechanism between the players?
5. Can the users understand the navigation between the different game states?
6. Are the users satisfied with the overall game?
7. Do the users require any training to understand the game?
8. What are the suggestions/recommendations of the users to improve the game?

Palm devices installed with the game are given to players in the group and they were asked to start the game. Each group was observed while playing the game. Once the player was done with the game, he/she was asked to update the score.

## **Results**

The results of the Usability Testing are explained below:

1. Players said that they were very happy with the option of choosing player type because if they get bored with one type of deck, they have a choice for getting a new type of deck with new cards having different levels of difficulty.
2. Players are happy with the system feedback, which is helpful to navigate from one state to the other state.
3. Players are impressed with the attacking and blocking state, which is very interesting and strategic.
4. Players are satisfied with the choices of communication - Bluetooth and IR beaming.
5. Players are excited with the display of top five players scores from the centralized database. They said it would increase friendly competition among them.

6. One of the players suggested that he needs some instructions once he reached the beam state. He did not understand what is going to happen when he reached the beam state. After we gave instructions, he continued the game without confusion.
7. One player suggested that animations would give more look and feel to the interface. We said that since the game itself occupied more memory, we do not have enough memory to provide animations.
8. One player suggested that it is hard to understand the game for the first time. Even though, it is a tough game one can follow after reading the rules in the help menu.
9. One player asked that he wanted to see opponent's playing cards. Because of the Palm's small screen area we are showing opponent's cards only in the most important phase of the game, the attack phase.

Table 4: Usability Testing Results

User Group	Expertise Level	Time Taken for single game	Comments
User Group 1	Extensive	45min	<ol style="list-style-type: none"> <li>1. Started without any confusion.</li> <li>2. Navigated between the states easily.</li> <li>3. Had problem with finding the Menu bar to select the menu options. Because the forms title hides the menu bar.</li> <li>4. Went through the attacking and blocking state.</li> <li>5. Once he reached a high score, he successfully quit the game.</li> </ol>
User Group2	Average	60min	<ol style="list-style-type: none"> <li>1.Started the game without any</li> </ol>

			confusion. 2. Navigated between the states easily. 3. Problem with finding the Menu bar to select the menu options because the forms title hides the menu bar. 3. Problem with attack and block state.
User Group 3	Minimal	80 min	1. Started the game without any confusion. 3. Navigation between the states is difficult. 3. Problem with finding the Menu bar to select the menu options because the forms title hides the menu bar. 3. Problem with attack and block state.

Source: Author

Average time for playing the game is approximately 60 minutes.

Most of the players were happy with the game design. Especially, they were very impressed with accessing top five scores dynamically from the centralized database. Players are happy with navigation between states and moves in attack and block state. They suggested that the interface would look very impressive if animations are provided.

# Chapter 6

## Conclusion and Future Enhancements

As discussed in the above chapters, a *Distributed System* is implemented as an integration of different platforms on different devices. The components implemented in the distributed system are the following: Palm Maya game on a Palm device, Game Conduit, Game Server, and centralized Oracle database on a desktop. As XML has the power to store, carry, and exchange data between different platforms, it is chosen to communicate data between all components in the distributed system. Oracle9i feature of *XMLType* tables and columns support the storing and retrieving of XML data to and from the database. The Game Server supports the multi-player environment that can be used to maintain consistent state between centralized database and game players. Game Conduit transfers large amount of data between Palm device and Game Server. Palm Maya game is implemented on the Palm device using the features of Palm OS like User Interfaces, Databases, Memory handling, Communications and Networking.

We have achieved the communication between the Palm devices and the Game Server by implementing the Multi-channeling pattern. Multi-channeling pattern was implemented to provide communication between the Game Server and Palm device in two ways: using conduit, and through Internet (using a mobile phone and Palm Net Library). The conduit is implemented to transfer large amounts of data between the Game Server and the Palm device. In our game, we have used the conduit for storing a deck of cards in a Palm



database and for storing the player's updated score into the centralized database. These two operations are not dynamic, as they do not interrupt the game. Palm Net Library was used to retrieve the top five scores from the database dynamically while the game is in progress. Even though the Palm Net Library supports transferring large amount of data; we delegated that responsibility to the conduit to reduce network traffic.

We have accomplished smooth navigation between the different game states by implementing the State-based pattern. The player needs to interact with the opponent once at the end of each turn during the game. But, the player in the attack state needs to communicate with the opponent more than once in a single turn. The communication between players was achieved using Bluetooth, and Infrared beaming.

To conclude, we have implemented a distributed gaming system by integrating technologies like Palm OS, Bluetooth, XML, Oracle9i, and Palm devices.

## **Future Enhancements**

In this project, we implemented a trivial signature scheme to maintain the game security. Later, one can plug-in a more sophisticated signature scheme to provide a higher level of security. The user interface of the game can be enhanced by providing more graphics and animations. In our game, each animal card has only four properties based on which the players play with each other during the attack and block phase. We can add more properties to the animal card to increase the toughness of the game.

# References

[ADGKR] Amund, A., Davide, B., Giuseppe, M., Kyle, B., Robert, H. “*Patterns of three-tier client-server architectures*”. <http://members.aol.com/kgb1001001/Articles/threetier/threetier.htm>.

[EJ] Eelke, F., Jan, B. “*Architecturally Sensitive Usability Patterns*”.

<http://www.eelke.com/research/vplop.pdf>

[EN00] Elmasri, R. A., & Navathe, S. B. (2000). “*Fundamentals of Database Systems*” (3rd ed.). Addison-Wesley.

[F00] Foster, L.R. (2000). “*Palm OS Programming Bible*”. Hungry Minds, Inc.

[HM02] Harold, R. E., & Means, S.W. (2002). “*XML in a Nutshell (2nd ed.)*”. O'Reilly.

[JS] Jussi, H., Staffan, B. “*Game Design Patterns*”.

<http://civ.idc.cs.chalmers.se/publications/2003/gamedesignpatterns.pdf>

[KBGP01] Kaljuvee, O., Buyukkokten, O., Garcia-Molina, H., Paepcke, A. (2001, April). “*Efficient Web Form Entry on PDAs.*” Proceedings of the tenth international conference on World Wide Web, 663-672.

[LLF98] Leventhal, M., Lewis, D., & Fuchs, M. (1998). “*Designing XML Internet applications.*” PrenticeHall.

[MSK03] Megowan, P., Suvak, D., Kogan, D. (2003). “*Infrared Data Association. (IrDA) Object Exchange Protocol (OBEX.)*”. Extended Systems, Inc Microsoft Corporation.

[ORA9i] *Oracle9i XML Database Developer's Guide* - Oracle XML DB.

[RG02] Ramakrishnan, R., & Gehrke, J. (2002). Database Management Systems (3rd ed.). McGraw-Hill.

[SM01] Sundaresan, N., Moussa, R. (2001,May). “*Algorithms and Programming Models for Efficient Representation of XML for Internet Applications.*” Proceedings of the tenth international conference on World Wide Web, 366-375.

[S01] Suciu, D. (2001,September). “*On Database Theory and XML.*” SIGMOD Record (8), 39-45.

[TVBSSZ02] Tatarinov, I., Viglas, D.S., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C. (2002,June). “*Storing and Querying Ordered XML using a Relational Database System.*” Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 204-215.