MODELLING PERSONALIZATIONS IN THE DESIGN OF MOBILE PUBLISH/SUBSCRIBE ARCHITECTURAL FRAMEWORK FOR SOUTH AFRICAN NATIONAL PARKS.

Mcebo Brendon Linda

(012298)

A dissertation submitted in fulfilment of the requirements for the degree of

Master of Science (Computer Science)

Department of Computer Science, Faculty of Science and Agriculture, University of Zululand

2006

DECLARATION

I, Mcebo Brendon Linda declare that this dissertation represents research work carried out by myself and that it has not been submitted in any form for another degree at any university or higher learning institution. All information used from published or unpublished work of others has been acknowledged in the text.

_____

Signature of Student

DEDICATION

I dedicate this work to my mother, Emmelinah Nogogo Linda, for understanding, believing and encouraging me to excel in education in order to bear her name since she never had the opportunity to do so. The work is also dedicated to my siblings (brothers and sisters), especially my young sister Thulile "Tina" and my mother's last born Ntuthuko "Mfana" as they still have a chance to follow in my footsteps by instilling the value of education in our family.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to all people who supported me throughout my research.

In particular, I offer my sincerest gratitude to my supervisor, Prof. M.O Adigun. He introduced me to research and kept me on the right track with his advice, while giving me the freedom and the possibility to pursue my research ideas.

I especially wish to thank Dr. Xulu for his continuous fatherly support and mostly for believing in me when the challenge became tough.

It is also my pleasure to acknowledge the great collaboration of our Department with the following research team, Dr. Adeoromu, Dr Justice Emuoyobafarhe, Dr. Ojo and Mr. Johnson Iyilade. They really made a great contribution to the completion of my work.

To my colleagues and friends, Edgar Jembere and Pragasen Mudali, I owe many thanks for our frequent meetings, your generous help and considerable input that led to the success of this thesis. I would also like to thank Mr. Klaas Kabini for his time that he dedicated in order to solve the problems that I encountered to make the development of the prototype a reality.

Besides the above mentioned people, I thank all staff members and fellow colleagues in the Department of Computer Science at the University of Zululand for their continuous encouragement and support.

To my mother Emmelinah and my siblings, Thulile "Tina" Hazel, Ntuthuko 'Mfana' Cyril, my twin sister Cebisile Brenda Linda, I am eternally grateful for their endless love and support. Finally, my entire family, has been a constant inspiration in my life and an infinite source of energy and motivation.

Finally, I would like to thank Telkom Company for funding my studies. Without their financial support the completion of this work would not be a success.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The dissertation presents the adoption of publish/subscribe pattern as the existing communication paradigm into the South African National Parks (SANPARKS) organization. It focuses on restructuring the existing SANPARKS system into a publish/subscribe information provider that would enable clients to receive personalized information services delivered via sms, mms and email. A publish/subscribe architectural framework, which supports both mobile and desktop users with the following features was developed: a national park system restructured into an information provider; a portal interface for information consumers; a mobile interface support achieved by personalization and a highly rated usability index. This research was conducted by firstly, surveying the theory which consisted of the literature review. Secondly, the formulative aspect which consisted of model building, the proof of concept such as prototype and the usability testing of the prototype. The results of this research testify that the newly restructured SANPARKS meet the standard of the information provider entity. It also provides a portal interface for information consumers enabled by personalization. From the results obtained in this research the adopted publish/subscribe pattern solves the problem of restructuring the SANPARKS system into an information provider that enables clients to receive personalized information services delivered via sms, mms and email.

CHAPTER ONE

INTRODUCTION AND BACKGROUND

1.1 Preamble

Tourism is a key component of Government's macro-economic strategy to achieve growth, employment and redistribution of wealth in South Africa. National parks are the cornerstone of nature conservation based tourism industry in South Africa. They offer visitors the very best of leisure opportunities including game viewing, bush walks, canoeing and exposure to cultural and historical experiences. National parks help preserve history and the natural beauty of the nation for the benefit of all citizens and international visitors. Parks are places for recreation and education and need to be taken care of in a way that preserves them for future usage. Conserving the natural wealth of national parks can only be accomplished with the continuous support and involvement of visitors and local communities. A one-stop information services system of South African national parks that simplifies information access to users through personalization has a great potential in securing customer loyalty and in capturing the interest of potential clients.

South African National Parks (SANPARKS) is the South African organization that maintains the national parks information services. There are many national parks in South Africa who are not part of the SANPARKS. In order for the national parks to be part of the SANPARKS they need to register with this organization. Currently the SANPARKS information systems are distributed and hence the need for a one-stop information

provider system that would ensure that information is simplified and customised through personalization. The current on-line SANPARKS information system does not support personalization. Personalization is about tailoring products and services to better fit the user, e.g. *by focusing on the user needs, preferences, interests, expertise, workloads, task etc* (www.sanparks.org). The focus of this study is to redesign the current SANPARKS Information System such that personalization is explored; thereby enriching the users' experience.

There are two widely used information access communication technologies, which are the pull and push technologies. The main advantages of the publish/subscribe system that supports push over pull communication technology are (1) It provides a platform for personalisation of the services to be accessed by the users, (2) users do not need to use the small keypad to access the information as in the pull technology, since the information will be pushed to the user when available. The proposed system will add value to the SANPARKS through the use of push style of information services combined with personalized messages. The achieved relevance and ease of use will increase the number of subscribers to SANPARKS because users will receive information easily through their mobile devices and desktops via sms, mms and emails.

Information access can be made more efficient by introducing a publish/subscribe system, which includes both the user's and provider's efforts as it supports push communication. Publish/Subscribe systems ensure that information is pushed to the users according to their preferences.

1.2 Background

Mobile commerce is a subset of e-Commerce where wireless digital devices are used to initiate transactions on the web and it offers new opportunities both for mobile device manufacturers and service providers (Zhang, et al., 2002; Pashtan, 2004). M-Commerce differs partially from e-commerce due to the special characteristics and constraints of the mobile devices and wireless network. The volume of mobile devices is continuously and increasingly driven by people who need to have ubiquitous access to services, information and entertainment. Access to these can only be made available through their mobile devices (Ozen, et al., 2004; Hueng, et al., 2004). With the increasing popularity of mobile handheld devices, there is a pressing need to extend publish/subscribe services to a mobile environment such as mobile commerce (Eugster, et al., 2003; Muhl, et al., 2004). A scalable information delivery service, which publishes and subscribes to systems, provide, is required to connect together information providers and consumers by delivering events from sources to interested users (Hueng, et al., 2004). Many users of and visitors to South African National Parks carry their mobile communication devices. Hence, mobile devices can become access mechanism to information services of the SANPARKS.

Currently there are 20 national parks in the country that are part of the SANPARKS organization. These are Addo Elephant, Agulhas, Augrabies Falls, Bontebok, Golden Gate Highlands, Karoo, Kgalagadi, Knysna, Kruger, Mapungubwe, Marakele, Mountain Zebra, Namaqua, Richtersveld, Table Mountain, Tankwa Karoo, Tsitsikamma, Vaalbos,

West coast and Wilderness (www.sanparks.org). The major activity of SANPARKS includes but is not limited to the following: firstly, a repository of nature conservation data that can be used by researchers, conservationists and tourists. Secondly, it integrates multiple repositories to form a single infrastructure for all the national parks. The mission of the SANPARKS is to transform an established system for managing the natural environment into one which encompasses cultural resources, and which engages all sections of the community (www.sanparks.org).

The activities of SANPARKS are broadly categorized by services and consumers into three categories namely: tourists, conservationists and researchers (see Table1.1).

Table 1. 1: Information provided by SANPARKS and respective Consumers (Khumalo, 2004)

	INFORMATION PROVIDED	CONSUMERS	DELIVERY MECHANISM
1.	Plant	Researchers, Conservationist	SANPARKS are still inhibited by legacies of the past with the use of static web pages that were designed to inform public about services or products offerings.
2.	Animal	Researchers, Conservationist	
3.	Features (Scenic and Cultural).	Researchers, Tourists	
4.	Accommodation	Tourists, Researchers	
5.	Archives	Tourists, Conservationists and Researchers	

1.3 Statement of the problem

The operational activities of SANPARKS consist of nature conservation, information service provision and delivery of available services to interested parties. An investigation of the current operational structure identified the need to address a number of issues:

First, the South African National Parks system can be enhanced beyond its present capability such that all the communities of South Africa can benefit from its activities. Secondly, the era of using static web pages previously designed to inform the public about services or products offerings should come to an end.

The issues above are addressed in this research by (i) restructuring the SANPARKS system into an information-provider entity. (ii) creating a corresponding publish/subscribe model and thereby producing valuable enough information that could be sold to information consumers; and implementing a prototype of the model that emphasizes personalisation, and thus make it possible for customers to be rewarded in future. It is necessary to generate information that is so valuable that consumers are willing to buy.

1.4 Motivation for the study

This research study is meant to provide a one-stop SANPARKS information system for nature conservation. Such a system would ensure easy access to information being a one-stop portal interface to the distributed SANPARKS information systems. Incorporation of

publish/subscribe techniques in the system would include personalisation of information to be accessed by mobile users. Given the fact that the mobile environment is characterized by poor power supply, a poor user interface and flaky network connection, personalisation will simplify information access under these conditions. Pushing of information according to the user's interests would subsequently add value to the SANPARKS that we have today. This would be achieved by reusing and extending related systems where possible to suite the need of the investigation.

Information and notification services for communicating time-sensitive data have proved their usability in the Internet domain. The huge success of Short Message Service (SMS) and the increased acceptance of Multimedia Message Service (MMS) advocate the extension of the initial application domain to mobile environments, and encourage further efforts to implement and deploy content dissemination services in mobile environments. However, mobile scenarios introduce additional requirements regarding the service: Mobile users want to be served with relevant and personalized content in a timely manner. Moreover, the content must be customized to their current presence status, and directed to the terminal they are currently using. Therefore, service flexibility and its ability to deliver personalized content that provokes no nuisance to end users is of major importance for the wide acceptance of the service. Support for personal mobility is needed to assure timely information dissemination in accordance with the user's present status.

1.5 Research Goal and Objectives

1.5.1 Research Goal

The focus of this study was to restructure the existing SANPARKS system into a publish/subscribe information provider that would enable clients to receive personalized information services delivered via sms, mms and email.

1.5.2 Research Objectives

The objective of this study was to develop a Publish/Subscribe architectural framework that supports mobile and desktop users and has the following features: A national park system restructured into an information provider; A portal interface for information consumers; A mobile interface support achieved by personalization and a highly rated usability index.

1.6 Research Methodology

The research approach is both theoretical and formulative in nature, consisting of literature review, model building and the proof of concept such as prototype implementation. The details are given below.

1.6.1 The Theoretical Aspect- Literature Review

The theoretical aspect of this research involves literature review. An investigation of existing information systems for SANPARKS, sources of information on nature conservation, tourism and publish/subscribe, personalisation and portal user interface strategies for mobile environment was carried out.

1.6.2 The Formulative Aspect

The knowledge gained from the literature survey was used to construct a theoretical background of the formulative part of this research. The formulative part of this research involved model formulation and proof of concept through software based conceptual analysis and implementation of the prototype.

1.6.2.1 Model Formulation

This involved formulating requirements to fit the following purpose: A distributed SANPARK information system; a publish/subscribe engine that drives the system; and a user interface that supports personalized mobile users. Existing patterns were used as the building blocks for creating the core publish-subscribe model from which the information system evolved.

1.6.2.2 Model Implementation

A prototype of the proposed model was implemented as a proof of the concept using Java as the programming language to demonstrate the applicability of our model. The prototype was then evaluated against the models from which it was developed and other closely related work such as (Khumalo, 2004).

1.7 The Structure of the dissertation.

The remainder of this dissertation is structured as follows: Chapter two gives an overview of existing solution and systems related to content dissemination in publish/subscribe systems: In this chapter the author analyses and compares the characteristics of prominent publish/subscribe systems, and related solutions, such as SMS, MMS, E-mails and push systems. Chapter three introduces the proposed system model of publish/subscribe systems that is used as the solution to current problems of the SANPARKS in order to improve dissemination of notifications to mobile subscribers and desktops subscribers.

Chapter four presents the computational implementation of the proposed Publish/Subscribe Architectural Framework and the implementation results. This chapter also gives the detailed usability analysis of implemented prototype. Finally chapter five presents the conclusion which consists of how the research objectives were achieved and possible future work for further research and results.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The publish/subscribe model represents an emerging paradigm for de-coupled and asynchronous connections between application components (publishers and subscribers). In other words, a publish/subscribe model may support two types of models which are mediator and implicit models as these models support decoupling and asynchronous connections between application components in contrast to peer-to-peer model only support coupling and synchronous connection between application components. The mediator and implicit models form the basis foundation of the architecture of our research. This research focuses on adopting the publish/subscribe communication model that allows de-coupled and asynchronous connection between publishers and subscribers. The adoption of publish/subscribe communication paradigm allows the SANPARKS organization to integrate the publish/subscribe engine where the information consumer get information without knowing the source.

SANPARKS is organized in such a way that it has to deal with publishing information on plants, animals, features (scenic and cultural) and accommodation. Subscribers or clients need to be notified when events that represent information items and published content occur. This model is event–driven because the act of publishing is periodic and guided by the availability of new or modified information item, or by a publisher's state change.

Publishers produce the information and subsequently publish it for dissemination to interested subscribers: Publishers are notification producers, while subscribers act as notification consumers. Notification must be preceded by declaring an interest in receiving specific categories (plants, animals, features and accommodation) of notification. When a notification is published, it will be delivered specifically to all registered subscribers. Publishers and subscribers may optionally interact directly by adopting peer-to-peer model but this is not supported by the SANPARKS model.

The mediator model introduces an intermediary, an "information bus" responsible for efficient notification delivery from publishers to subscribers. The intermediary ensures the anonymity of the communicating parties: Publishers and subscribers do not necessarily need to know each other and the infrastructure keeps track of subscriptions and publications. Furthermore, the interaction style enables one-to-many multicast-style communication such that a published notification is delivered to all interested subscribers.

The implicit model introduces the concept of intermediary, an "information bus" between the subscribers and the publishers. The subscribers register for the types of topics when one would like to receive notification when it becomes available to the system. Publishers are responsible for publishing information that is related to the available topics on the system.

Most publish/subscribe systems adopt either mediator or implicit models. This is mainly because these models, as discussed above, avoid the need for a subscriber to know the

publisher of the information subscribed for and vice versa, hence the delivery of notifications are guaranteed even if the subscribers are disconnected from the network.

Systems that implement the publish/subscribe interaction style usually fall the category of software infrastructure built on top of the network operating system, that offers generic services for the development of distributed applications. The main purpose of software infrastructure of this nature in practice is to simplify the implementation of distributed systems and is referred to as middleware (Emmerirc, 2000). In this research work, the middleware played a major role in simplifying the implementation of the envisaged SANPARKS Information System which was a one-stop information provider system. Publish/subscribe systems are often classified as event-based middleware because of the event-driven communication and cooperation model that uses notification for carrying the information that is passed among the communicating parties (Meier, 2000). Tanenbaum (2002) classifies publish/subscribe as coordination systems to stress that the publish/subscribe interaction coordinates the activities between distributed processes.

2.2 Analysis of Publish/Subscribe Systems

Our analysis of publish/subscribe system study identifies that publish/subscribe historically are variants or a combination of the following communication styles: Peer-to-peer/listener model; broker (mediator) model and implicit model.

2.2.1 Peer-to-peer Model

In the **peer-to-peer model,** subscribers register at specifically named publishing entities and the publishing entities deliver events to specific named subscribing entities directly. The interaction style enables one-to-one communication because the published notification is delivered to one interested subscriber that requests the notification from the publisher which acts as the server in that particular time. The communicating parties of this model have the knowledge of each other since they hold the reference of each other as they are communicating.

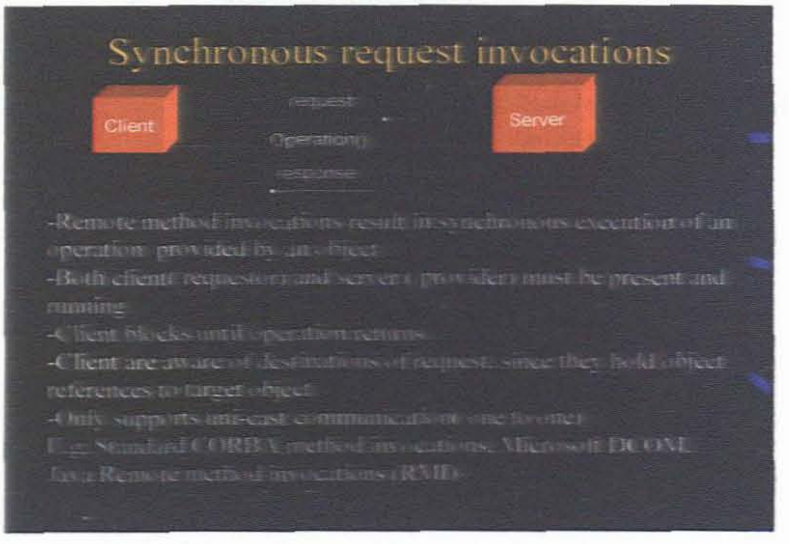

Figure 2. 1: Peer-to-peer Model for Publish/Subscribe Systems (Fiege, et al., 2003).

2.2.2 Broker (Mediator) Model

In the **broker (mediator) model** the subscribers register their subscriptions with a common event mediator which introduces an intermediary. The intermediary ensures the anonymity of the communicating parties: publishers and subscribers do not necessarily

need to know each other and the infrastructure keeps track of their subscriptions and publications. Furthermore, the interaction style enables one-to-many multicast-style communication because the published notification is delivered to all interested subscribers. The Publisher forwards events to the common event mediator, who in turn takes care of receiving events from publishers and delivers them to all interested subscribers.

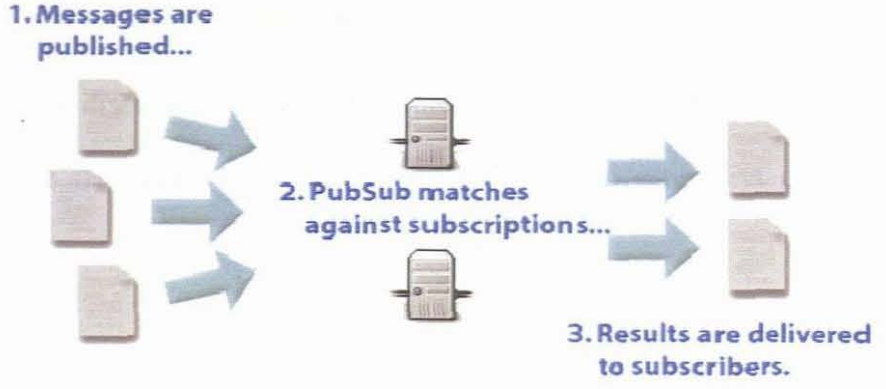

Figure 2. 2: Mediator (Broker) Model for Publish/Subscribe Systems (Eugster, et al., 2003).

2.2.3 Implicit Model

The **implicit model** requires the subscribing entities to subscribe to (or register) a particular event type rather than with a mediator or publishing entity e.g.; in figure 2.3, S represents subscribers, therefore S needs to subscribe to a Topic as the event type for the purpose of receiving the related topics from P who is the Publisher. Consequently, the publishing entities generate events of some type, which are delivered to subscribing entities. Finally publishers or subscribers need not hold explicit reference to mediators or publish/subscribe entities (Quah, et al., 2002).

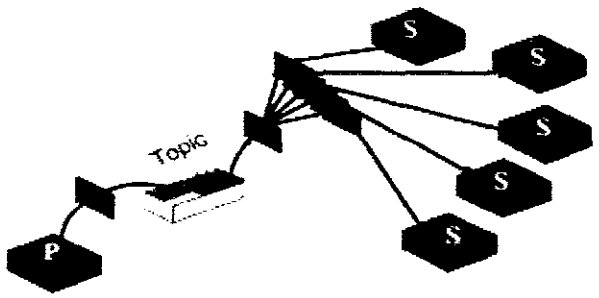

Figure 2. 3: Implicit Model for Publish/Subscribe Systems (Quah, et al., 2002).

2.3 Representative Publish/Subscribe Systems

This section discusses the different kinds of existing publish/subscribe systems that demonstrate the features set out by the concept and categories of publish/subscribe models.

2.3.1 CORBA Event and Notification

The CORBA Event Service: The CORBA Event Service provides a decoupled communication model that addresses the limitations with the CORBA SMI (and AMI) invocation mechanisms outlined above. As shown in Figure 2.4, the Event Service defines three roles:

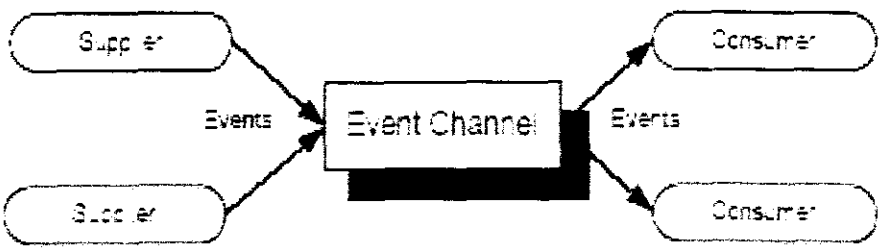

Figure 2. 4: Components in the CORBA Event Service (Object Management Group, 2001)

- *Suppliers*, which produce event data;
- *Consumers*, which receive and process event data;
- *Event channel*, which is the mediator through which multiple consumers and suppliers communicate asynchronously.

This type of publish/subscribe system is based on the mediator model since it uses the mediator as the "information bus" between the suppliers and consumers to avoid the prior knowledge of each other as they exchange notifications. The communication style between the suppliers and consumers is multicasting because many suppliers produce event notifications consumed by many consumers just like in a publish/subscribe communication paradigm.

In a simple scenario consumers and suppliers can interact directly, without a channel, by invoking each other's interface methods. A mediated scenario involves a channel that acts as both a supplier and a consumer of events. Event channels enable anonymous many-to-many communication between suppliers and consumers. However, event channels offer no means for event filtering: All channel consumers receive all events published on the channel.

CORBA event service supports both the *push* and *pull* approaches to communication initiation: The push model allows suppliers to initiate the distribution of notifications to consumers. The pull model allows consumers to request notifications from suppliers. Push suppliers actively send notifications to the event channel, while pull suppliers wait for requests coming from the channel. Push consumers passively wait for events that are

eventually sent through the channel, while pull consumers regularly check if new events are available on the channel.

The event service defines simple means for event propagation and has a number of drawbacks. It is not adequate for mobile scenarios because consumers must be connected to the channel at the time of event publication. CORBA event service does not support event persistence. The second drawback is that event channels offer no means for event filtering. If event types are to be differentiated, it is necessary to use separate channels for each event type. Finally, the specification does not dictate the reliability requirements for the communication service and offers no guarantees concerning the delivery of events. It can have either "at-most-once" or "exactly once" semantics, depending on the particular service implementation.

Notification service: CORBA notification service (Object Management Group, 2002) deals with the above-mentioned drawbacks of the event service and extends it with new capabilities, such as filtering and configurability, according to various requirements for quality of service (QoS). The notification service preserves the semantics of the event service and ensures interoperability between the basic event service clients and notification service clients. One of the extensions offered by the notification service are content-based subscriptions and event filtering using filter objects: filter objects define a set of constraints that affect the forwarding of an event. For example, notification service consumer can subscribe to events of interest by associating a filter object to the proxy through which it connects to an event channel. When an event that matches the filter

object is published, the proxy will forward it to the consumer. This implies that the notification services supports personalization of services when it delivers information from the supplier to the consumers as described on the example given above.

The notification service introduces a new type of events, *structured events* with a well-known data structure into which a wide variety of event types can be mapped. Structured events consist of a header and a body: A *header* is further decomposed into a fixed and a variable part. The fixed event header consists of a domain_name which identifies a particular domain (e.g. telecommunications, finance), a type_name which categorizes an event, and an event_name which can uniquely specify an event. A variable header part is composed of a list of optional name-value pairs. *Event body* carries the content of an event. The filterable portion of the body contains the most interesting event fields (name-value pairs) used when matching the event with a filter object. The remainder of the body is of type any and can be used to transmit large data items.

Another enhancement introduced by the notification service are standard interfaces for controlling QoS characteristics for event delivery. The notification service enables each channel, each connection, and each message to be configured so as to support the desired quality of service with respect to delivery guarantees, event persistence, and event prioritization. OMG defines a set of QoS properties, their permitted types, and the range of values. This is an open list of parameters, and service implementers can add their own properties. OMG has defined the following properties:

• Reliability is related to the event delivery policy, such as best effort, or persistent delivery;

• Priority; by default, the notification channel will attempt to deliver messages according to their priority level;

• Expiry times indicate the time interval within which an event is valid;

• Earliest delivery time specifies the time after which an event can be delivered and

• Maximum events per consumer define the maximum number of events a channel can queue on behalf of a consumer. This property prevents malicious users from overloading a channel.

The list of supported properties provides flexible QoS configuration of a notification channel. However, meaningless properties are not prevented which creates a serious vulnerability that could be exploited by malicious consumers or suppliers. End-to-end delivery policy can only be guaranteed with the cooperation of all parties, i.e., consumers, suppliers, and the notification channel. The OMG event and notification service specifications offer no guidelines regarding the architecture and routing strategy for distributed event systems.

2.3.2 Java Message Service

This type of publish/subscribe system is based in all the three types of publish/subscribe models which are peer-to-peer, implicit and mediator model. Java Message Service (JMS) is a message-oriented specification for the Java programming language defining a set of interfaces and semantics, thus enabling JMS compliant clients to access the

services offered by a JMS messaging server. The JMS target application area is enterprise messaging for asynchronous Business-to-Business communication over the Internet. JMS provides two types of messaging models, point-to-point messaging and publish/subscribe (Monson-Haefel, et al., 2001). The point-to-point messaging model relies on the classical message-queuing communication pattern that make this system to be based on the peer-to-peer model because as the server respond to clients which are on queue actively waiting for the information to be delivered to them is active. JMS system uses point to point messaging to deliver information to interested parties. The reason for the JMS system to be based on the implicit model is because it uses topics to represent the type of event in which the consumers can subscribe for and wait for the information that is relevant to the topics of their interest to be available before it can be pushed to the matching subscribers.

This system uses both pull and push communication technology since it allows the subscribers to pull information that is related to the topics of their choice, subscribers have an option of waiting for the information that match their subscription to be pushed to them by the publishers when it becomes available. Publishers publish messages to a *JMS topic*, which is one of JMS destinations. Topics are created by an administrator using the administrative tools offered by the applied JMS provider. It is assumed that publishers will publish messages on the established topics. This approach is static and is augmented by *temporary topics*: Publishers can dynamically create new temporary topics. Subscribers subscribe to a particular topic by registering their message listeners

with the topic, as depicted in Figure 2.5. Whenever a message is published on a topic, the listener's method is invoked, signaling the receipt of a new message for the subscriber.

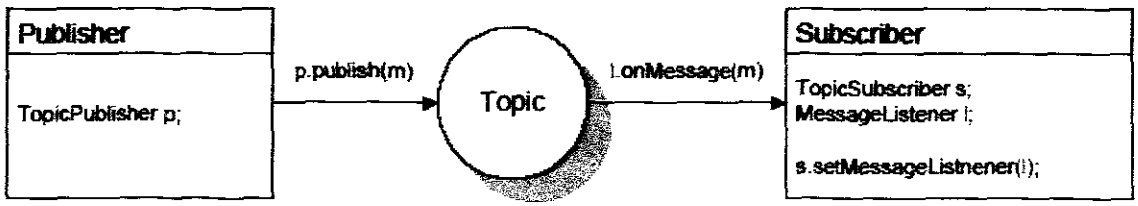

Figure 2. 5: Publish/subscribe interaction in JMS (Monson-Haefel, et al., 2001)

The JMS system is also based on the mediator model as it adopts the publish/subscribe model that incorporates two type of JMS clients, publishers and providers, that communicate by exchanging messages through an intermediary server, called JMS provider. A *JMS provider* is a messaging server that implements JMS interfaces and provides administrative and control features. *JMS clients* are programs or components written in the Java programming language that produces and consumes messages. Notifications are referred to as *messages* in JMS: Messages are Java objects that communicate information between JMS clients.

JMS offers delivery guarantees using the concepts of *durable subscriptions* and *persistent messages*. Subscribers can define durable subscriptions to a topic. While a durable subscriber is disconnected from a JMS server, the server stores the published messages matching its subscription. When the subscriber reconnects, the server sends all stored and unexpired messages to the subscriber in the *store-and-forward* delivery style. Publishers can define either persistent or non-persistent delivery mode for their messages. In the

case of the non-persistent mode, the server offers at-most-once message delivery. Persistent messages are first stored by the server, and then delivered to subscribers. Subscribers need to confirm the receipt of a persistent message. If the acknowledgment is missing, the server resends the message assuring at-least-once message delivery. JMS defines *message filtering* on the subscriber side using message selectors. Message selectors are expressed as Java strings that define conditions on message properties and headers. Message selectors need to comply with the defined subscription grammar which supports the conditions as complex boolean expressions with equality, comparison, or range operators. The JMS specification is a pure API specification. It does not define the rules for building the architecture of JMS server with respect to distribution.

2.3.3 JEDI

The Java Event-based Distributed Infrastructure (JEDI) (Cugola, et al., 2001) is a lightweight middleware infrastructure that supports the development of event-based application. This system is based on the mediator model as it uses the types of events, in which the subscribers subscribe for in order to gain access of information stored by the events. JEDI is based on the concept of Active Objects (AO) and Event Dispatchers (EDs). An AO is a special kind of object that interacts with other AOs by producing and consuming events. An AO can thus perform the activities of both an event publisher and a subscriber to a particular event type. An ED is a special component responsible for delivering events from publishing AOs to AOs that have expressed interest to receive such events.

2.3.3.1 Jedi Event and Event Patterns

A JEDI event is an ordered set of attributes that describes event characteristic. An attribute is a name-value pair. Both name and value are strings and, as a consequence, an event is a sequence of strings. JEDI supports content-based event filtering that appliers pattern matching based on regular expressions when comparing event to subscriptions. An AO can either subscribe to a specific event, or to an event pattern.

Event patterns are ordered set of strings that represent a simple form of regular expressions over events. An event pattern is identified with a sequence of pairs (name, regular expression), where name and regular expression are both strings. A pattern-matching algorithm is used to verify the compatibility between an event instance and an event pattern. For example, the event pattern (Source_ID, 12*); (Signal_Type,*) is compatible with all events with a value for attribute Source_ID starting with 12, and with any value of attribute Signal-Type.

An ED stores all event patterns received from subscribing AOs. When the ED receives an event, it verifies the compatibility between the received event and each event pattern using the pattern-matching algorithm, and delivers the event to each AO with the matching subscription that is connected to it.

2.3.3.2 Distributed ED Architecture

An ED is a logically centralized component that needs to have a global knowledge of AOs, their subscriptions, and published events. However, the centralized implementation of the ED is a critical bottleneck for a distributed system. To solve the scalability

problem, JEDI offers a distributed implementation of the ED. The distributed version of the ED consists of a set of dispatching servers (DSs). DSs are connected to form a tree topology. Each DS is located on a different network node and is connected to one parent DS and to zero or more descendant DSs. A DS with no parent DS is the root of the tree, while DSs with no descendant DSs are the leaves of the tree. AOs can connect to all DSs that form the ED.

DSs use a coordination protocol that distributes the information about subscriptions and events among them. The distribution protocol is designed to minimize the network load generated by control messages exchanged among the DSs. JEDI uses the hierarchical strategy to distribute events, subscriptions, and unsubscription messages between DSs: Subscriptions are propagated upwards in the tree, so that all ancestors of a DS receive it; when a DS receives a new event, it needs to send it to its connected AOs with a matching event pattern, its descendant DSs that have subscribed with a matching pattern, and its parent. This strategy ensures that all relevant nodes and the connected AOs receive the published event messages. However, this strategy has a significant weakness since events are always sent upward to the root DS which may become a serious performance bottleneck.

2.3.4 Siena

Scalable Internet Event Notification Architecture (Siena) (Carzaniga, 2001) is a middleware infrastructure that supports the implementation of publish/subscribe-based applications with the main objective to provide a scalable Internet-scale notification service. Siena is implemented as a distributed overlay network of servers that provide

clients with access points to a publish/subscribe interface. Notifications are produced by objects of interest and consumed by interested parties. Siena offers an advertisement mechanism that enables objects of interest to announce the type of notifications they intend to publish. Interested parties subscribe to notifications by defining an event pattern that serves their interests. The event pattern that has been defined by the interested parties makes this system to be based on the mediator model. Siena servers which act as mediators are responsible for selecting the notifications of interest and for delivering them to interested parties.

2.3.4.1 Notifications, Filters, and Patterns

Siena notifications are untyped set of typed attributes. Each attribute is a triple consisting of type, name, and value. A filter selects notifications by specifying attributes and constraints on the values of those attributes. Constraints are expressed using equality and ordering relations, substrings, prefix, and suffix operators for strings, ant the operator any matching any value. A filter is matched against a single notification based on the notification's attribute values. Additionally, Siena offers a limited support for composite events. It is possible to investigate a combination of notifications through the use of patterns. A pattern is defined as a sequence of filters that is matched against a temporally ordered sequence of notifications. For example, if two notifications are received one after the other and they match two filters that compose the client's pattern those notification will be delivered to the client.

2.3.4.2 Distributed Architecture

Siena is designed to offer scalable event distribution in wide area networks. A network of interconnected Siena servers builds the service infrastructure. Reference (Carzaniga, 2001) defines and analyzes four different server topologies: centralized, hierarchical, acyclic peer-to-peer, and general peer-to-peer. A control algorithm based on the principle of reverse path forwarding is applied in hierarchical and peer-to-peer topologies. The main idea behind the routing algorithm is to send notifications only to those servers that have clients interested in receiving such notifications. The algorithm is based on the principles found in IP multicast:

Downstream replication; a notification is routed in one copy as far as possible and replicated only downstream, as close as possible to the parties interested in it.

Upstream evaluation; filers are applied and assembled upstream, as close as possible to the sources of notification.

The forwarding of advertisements decreases the number of control messages that update subscription information since a subscription update is sent only to those servers who must generate the matching notifications. Advertisements set the routing path for subscription, which in turn set the path for notifications. Every advertisement is broadcast to all Siena servers. When a server receives a subscription, it propagates the subscription in the reverse direction, along the path to the advertiser, and activates the path for notification forwarding.

2.3.5 REBECA

The REBECA notification service (Fiege, 2003) is a content-based publish/subscribe infrastructure that consists of a set of interconnected brokers (mediators) that allow clients to publish notifications for interested users. Brokers are divided into two categories: Local brokers serve as access points for publisher and subscriber processes, while routers are used for forwarding messages between their neighbouring brokers.

A notification in REBECA is a message that contains information about an event that has occurred. A notification consists of a set of attributes where each attribute is a name-value pair. Notification filters are defined as boolean functions that can be applied to notifications. Filters can be either simple atomic predicates or compound filters. Simple atomic predicates contrast attributes to values using the operators such as equality, comparison, set operators, or string operators. A compound filter is a conjunction of simple filters.

The notification service is distributed and relies on a set of routing algorithms for delivering notifications: simple routing, identity-based routing, covering-based routing, and merging-based routing. All algorithms are based on the reverse path forwarding approach and can apply advertisements to avoid subscription flooding. In simple routing, all active filters are added to the broker routing tables with the identity of the link they originated from. This approach is not optimal because the routing tables grow linearly with the number of subscriptions. The straightforward improvement of the approach is to combine equal filters in routing tables, the approach used in identity-based routing.

Further improvement is the covering routing strategy which considers covering among filters to decrease the size of the routing tables. Finally, the most complex approach emerging that is to use to create new filters that cover existing filters (Muhl, 2001). The REBECA working prototype has been used to evaluate and compare the listed routing strategies in (Muhl, 2002).

2.3.6 Service-Based Architectural Framework for the South African National Park System

This system is based only on the peer-to-peer model because every time the subscriber accesses the SANPARKS services information he or she request the information stored on the static database. This system does not support the push communication technology in delivering the services information to the interested parties. This automatically declares that each time subscribers need to access the SANPARKS services have to place the request to the SANPARKS server. Generally this system supports client/server kind of communication because the subscriber acts as the client and the publisher acts as server.

The Service-Based Architectural Framework for South African National Parks system (Khumalo, 2004) does not support mobility when it delivers information to the subscribers. It only uses simple desktop interfaces to exchange information between the publisher and the subscriber. It also does not provide the related approach of delivering messages or information from the publisher to the subscriber. The related approaches that can be used to deliver notifications are SMS, MMS and Emails. The new proposed system in the last section of this chapter aims to improve the work that has been done by

(Khumalo, 2004) in order to provide the system subscribers with the personalized information about the SANPARKS services. The system designed in (Khumalo, 2004) supports both, the push and pull approaches to communication. The push model allows suppliers to initiate the distribution of notifications to consumers. The pull model allows consumers to request notifications from the suppliers. Push suppliers actively send notifications to the event channel, while pull suppliers wait for requests coming from the channel. Push consumers passively wait for events that are eventually sent through the channel, while pull consumers regularly check if new events are available on the channel.

This system only supports unicasting and also considers the issue of personalization when it delivers information about South African National Parks services to the subscribers. This system does not solve the issue of context awareness for mobility. The system (Khumalo, 2004), only cater for people who have subscribed to the system as customers of this system. It allows subscribers to unsubscribe from the system if they are no longer interested in receiving the services information from SANPARKS.

2.4 Mobility Support in Publish/Subscribe Systems

Most of the existing publish/subscribe systems have been designed and optimized for stationary environments where publishers and subscribers are static, and the infrastructure itself remains fixed. The mobility-related operation is dealt with at the application layer through a sequence of subscribe-unsubscribe-subscribe request. A subscriber from the application layer first defines new subscriptions and unsubscribes prior to disconnecting from the publish/subscribe system. After reconnecting to the

system, the subscriber needs to re-subscribe to make the system aware of its subscriptions. However, the subscriber will not receive notifications that have been published during the time of disconnection.

Podnar, (2002) argues that the publish/subscribe middleware itself must offer the mobility support by ensuring seamless reconnection to a new broker and by preserving notifications published during disconnection. Zeidler (2003) agrees that mobility-related issues should be addressed by the publish/subscribe middleware, and not delegated to the application layer. Some publish/subscribe systems address the problem of client mobility and disconnection by using a common solution of storing each notification published during disconnection in a special subscriber queue and deliver the notifications after subscriber reconnection. The existing solutions, in systems like Jedi, Siena, Jms and Rebeca extend the established stationary publish/subscribe systems to cope with client mobility while keeping the infrastructure stationary (Caporuscio, 2003). In the subsections below we look at the extent to which mobility is being supported in systems like Jedi, Siena, Jms and Rebeca.

2.4.1 Mobility Support in Jedi

JEDI (Cugola, 2001) offers two mobility-related operations: moveIn, and moveOut. A subscriber uses moveOut to disconnect from a broker and moveIn to reconnect possibly to a new broker. A client can detach from the system, serialize its current state, and later reconnect. The old broker stores events on behalf of the subscriber during the disconnection period and transmits them to a new broker upon reconnection. The

approach solves the queuing problem, however, no details regarding the handover procedure from the old to the new broker, or the change of the delivery path is given.

2.4.2 Mobility in Siena

The authors of Siena present a support service for mobile, wireless clients of a distributed publish/subscribe system in (Caporuscio, 2003). The mobility service enables the movement of subscribers between different access points of a publish/subscribe system. The service uses client proxies and a special client library to manage subscriptions and notifications on behalf of a subscriber, both while the subscriber is disconnected and during the handover between different access points. A client proxy runs as a special component at an access point and stores messages for a disconnected subscriber in a special queue. The client library mediates subscriptions, and initiates a move-out procedure. It submits subscriptions to the client proxy and submits the address of the old proxy. The old and new proxy start a special handover procedure that transfers messages from the old proxy to the new one and then to the subscriber.

The mobility service implements a special synchronization mechanism to avoid lost notifications. The main principle is quite simple: when transferring subscriptions from A in order to be active on B, the system needs to make sure that subscriptions are active on B before terminating subscriptions on A. It is possible that during the procedure both A and B will receive the same messages. The mobility service implementation permits that a subscriber receives duplicated messages. The presented system is independent from the underlying publish/subscribe middleware: The portability of the mobility service has

been proved through an implementation on top of three different publish/subscribe systems (Siena, JMS and Elvin). The client library wraps the targets publish/subscribe API and needs to be implemented specially for each API by adding the move-in and move-out functions, and by overriding the subscribe function of the original API.

Caporuscio (2003) gives the results of an experiment that shows the applicability of the implementation. The evaluation is limited since the experiment was performed on a broker network consisting of three broker nodes, a single publisher, and a single mobile subscriber that moves only once. The experiment included the performance evaluation if a subscriber uses a GPRS network which has been simulated to access the publish/subscribe service.

2.4.3 Mobility in Rebeca

The approach taken within the project REBECA is to extend and modify the existing publish/subscribe system to support mobile and location-dependent applications (Fiege, et al. 2003). The mobility service aims to support two different types of mobility: physical mobility and logical mobility. Physical mobility is similar to terminal mobility: A client is physically mobile and roams between different network domains. It can disconnect from the system and later on reconnect possibly to another broker in a different network. Its subscriptions are valid and the system stores notifications published during the disconnected period. Logical mobility is related to geographical location: as a client changes its geographical position, its subscriptions dynamically change because the published information is location-dependent.

The algorithm that is developed for physical mobility is designed for a distributed network of brokers. It applies the "queuing" approach: The old broker stores notifications for a disconnected subscriber. When the subscriber connects to a new broker, it re-issues its subscriptions, but keeps no record of the old broker address. The algorithm finds the old broker by locating a broker that is at the junction of delivery paths for the new and the old broker. It is clear how this junction broker is found if simple routing is used. Each broker stores active subscriptions for all subscribers with the subscriber identifier, and since the subscription from the old broker is still active in the system, it is simple to find the junction broker leading to both the old and the new broker. The notifications stored by the old broker are routed through the junction to the new broker and delivered to the subscriber. With simple routing the routing tables can become rather large because all brokers have to knowledge all subscriptions. Routing algorithms that use covering and merging are better suited for mobile environments where subscriptions change more often that in static scenarios. The proposed algorithm needs further extensions in case routing based on covering or merging is applied since the process of finding a broker junction is not straightforward. The designed algorithm appears to be rather complex and there are currently no evaluation results that shows its applicability and performance.

2.4.4 Mobility in JMS-Based Systems.

Recently, some of the systems that implement the JMS specification offer support for mobility (Yoneki, et al., 2003). Such systems offer a lightweight JMS compliant API for Java-enabled mobile terminals that can be used to implement JMS-based publishers and subscribers. iBus//Mobile is a commercial JMS-compliant implementation. It integrates a

special gateway that serves as a mediator between a JMS provider, and JMS clients. It offers support for native clients with no JMS support. Native clients can publish and receive SMS or MMS messages that are transformed into JMS messages that can interact with the JMS provider. iBus//Mobile supports TCP, UDP, HTTP, and HTTPS as transport protocols for JMS messages. JROM is an open source project that has recently published a client API called kJROM that adjusted to J2ME devices. Pronto (Yoneki, et al. 2003) is an academic project. It provides a JMS-compliant middleware system that supports mobility of JMS publishers and subscribers, and implements a mobile JMS API that can run on resource-limited devices. It incorporates a mobile gateway that supports JMS in wireless networks and employs SMS, or mail as transport mechanisms for native devices that do not support Java and JMS.

2.5 Related Technologies.

Notification is very essential to this research. Hence the need to survey technologies that can serve as candidate mechanism for notification. Those discussed in this section are electronic mail, sms, mms and push technologies.

2.5.1 Electronic Mail

Electronic mail is one of the first services on the Internet for distributing messages with arbitrary content. The introduction of mailing lists provides a powerful tool for one-to-many content dissemination. Tools for creating and maintaining mailing lists facilitate the users to subscribe to and unsubscribe from mailing lists automatically, and enables topic-based publish/subscribe interaction. The main disadvantage of using mail for

disseminating content to large mailing lists is resource consumption. The typical mail distribution method creates a separate mail copy for each receiver from the mailing list and sends each copy separately to the receiver even if several receivers use the same mail server. This approach can cause considerable computing load and bandwidth consumption which can lead to significant delivery delays.

Contrary to the huge success and primacy on the Internet, electronic mail is not currently widely used in the mobile domain. The main reasons for its poor acceptance are bandwidth limitations and scheduled pull-style retrieval of mail messages which require permanent network connection. Mail readers for mobile devices that apply standard Internet protocols (POP3 and IMAP) are currently available. To solve the problems related to the pull-style operation, proprietary solutions that employ push-style message retrieval have recently been developed. These solutions send notifications to the user's mobile terminal when a new mail message arrives at the mail server.

2.5.2 Short Message Service

Short Message Service (SMS) is a simple messaging service widely used in today's mobile networks (Le Bodic, 2003; Peersman, et al., 2000). An SMS transports alphanumeric messages using the store-and forward paradigm. Messages are temporarily stored if users cannot retrieve them at the time of message publication. A stored message is delivered to the user terminal when it reconnects to the network. An SMS offers a point-to-point service that enables person-to-person and machine-to-person message exchange carrying at most 140 bytes of payload, either 160 7-bit characters, or 140 8-bit

characters. In addition to the point-to-point communication, an SMS offers the so-called cell broadcast service for transmitting messages to all active terminals in a cell that have subscribed to the particular information service. This feature enables the deployment of information services carrying for example weather updates and financial reports that are examples of machine-to-person SMS usage scenarios. SMS is an extremely popular messaging service, but limited by the low bandwidth communication channels.

2.5.3 Multimedia Message Service

Multimedia Message Service (MMS) is an enhanced messaging service that exploits the access to higher bandwidth in 2.5G and 3G networks (Le Bodic, 2003). MMS enables the exchange of multimedia messages carrying text, audio, and pictures in the context of person-to-person and machine-to-person scenarios. MMS supports interoperability with electronic mail which gives rise to various usage scenarios. The concept of message notification allows deferred retrieval of messages and relies on persistent network-based storage of messages. Messages can be stored persistently in the network and controlled remotely via mobile terminals. Value-added services such as weather notifications, news updates, or location-based information are typical content dissemination applications that can be deployed using MMS as a transport mechanism. These services lack the flexibility of subscription found in publish/subscribe systems: The subscription to value-added services is static and currently offers no means for adjusting the service to user preferences and up-to-date needs. It would be useful to extend the MMS architecture which would then publish/subscribe interaction principles.

2.5.4 Push Technology

The Push technology offers timely delivery of possibly large amounts of content to many subscribers in wide area networks. The technology requires that channels are used to classify the content that is published to subscribers, and the term *push service* is used to stress that the content is actively pushed to subscribers. Push systems and publish/subscribe systems are closely related. The basic interaction model is the same. Subscribers subscribe to the service and receive the published content in the push style. The main difference between the two types of systems is that push systems offer services to end users, while publish/subscribe systems are middleware. Push systems offer channel-based subscription criteria to their users, while publish/subscribe systems provide flexible and expressive subscription capabilities. The extensive comparison of push systems and publish/subscribe middleware can be found in Minstrel (Hauswirth, 1999).

Minstrel is a Java-based push system developed at the Technical University of Vienna. The main goal is to provide flexible and secure content delivery in the area of e-commerce, and to ensure system scalability. Minstrel has a distributed architecture and employs a proprietary application-layer protocol for efficient content distribution to numerous users across a wide area network. The main Minstrel components are a broadcaster and a receiver. A broadcaster is responsible for managing channels and sending information along channels. A receiver component is responsible for subscribing a user to available channels and for receiving the content. The current receiver

implementation is designed for desktop computers, and the system does not support receiver mobility.

2.6 A Concise Overview of the Proposed Solution

In this dissertation a Publish/Subscribe Architectural Framework for SANPARKS is proposed for use by mobile and web-based applications. This research combines both the broker (mediator) model and implicit models into a design that encompasses the following criteria:

1. Restructuring – Restructure the SANPARK system fit for the status of information provider;
2. Transparent Information Delivery – Integrate a publish/subscribe engine that allows a consumer to get information without knowing the source;
3. Personalization – Enhance the engine to serve mobile users in a personalized manner.

The adoption of m-commerce in this research will help us in the definition of the architectural framework envisaged. From the point of view of access modalities, m-commerce services can be characterized as either subscribed or un-subscribed. Subscribed services are mostly used in both e-commerce and m-commerce, because they have stronger security level due to the personalization of services for specific users (Quah, et al., 2002; Ozen, et al., 2004). Un-subscribed services due to their time-limited nature always need more complex interaction between the user and the system, which implies a longer time to access the service.

This research work considers only subscribed access modalities because of the finite nature of services provided in the South African national parks. The work which has been done in (Posland, et al., 2001) and (Schmidt-Belz, et al., 2002) gave rise to a personalized, location-aware tourism support, implemented as a multi-agent system with the concept of service mediation and interaction facilitation. Personalization is one of the key features to facilitate the use of complex services on mobile devices (Barkhuus, et al., 2003). Personalization is where applications let the user specify his own settings for how the application should behave in a given situation (Yoneki, et al., 2003; Barkhuus, et al., 2003). We have architectural components needed to realize a personalization system.

These components include user profiles, where preferences are stored, and personalization rules that match user attributes and content (Pashtan, 2005). Finally the combination of localization and personalization would create a new channel/business opportunity for reaching and attracting customers. The system seeks to implement and trial tourism-related value-added services for nomadic users across mobile and fixed networks.

SMS and E-mail have been chosen as the delivery approach to subscribers since the aim is to publish SANPARKS services information to consumers in terms of text using both mobile devices and normal computers and these approaches are suitable for that. We then introduced personalization in order for the proposed solution to meet the standard of certifying user's preference. In order to achieve a comparable result with similar research

endeavours, the MVC pattern has been adopted in the design.The following subsections discuss the issues of personalization and mobility in the new system.

2.6.1 Personalization in A Design of Mobile Publish/Subscribe Architectural Framework.

A Mobile Publish/Subscribe Architectural Framework only supports the push approach to communication. The push model allows service suppliers to initiate the distribution of notification to consumers that already subscribe to the system in order to achieve the publish/subscribe paradigm. Services Suppliers which are known as Publishers in this system actively send notifications to the mediator for the mediator to check the matching subscribers and push the notifications to subscribers. In the push approach of communication Service Consumers passively wait for event notifications sent through the mediator, unlike in the pull approach where service consumers regularly check if new events are available at the mediator.

The model supports both unicasting and multicasting. It also enables personalization such that consumers receive information from South African National Parks services in a personalized manner. Personalization is the basic goal of this research work and without its achievement the research is not successful.

This solution uses the SMS (Short Message Service) and E-mail technologies to deliver the services information to the subscribers. This implies easy access to SANPARKS information services by the subscribers. In reality most of the people can not afford the

types of cell phones that support all the available delivery approaches that are available currently e.g. MMS and Push systems etc. Therefore SMS and Emails are the methods for delivering the information services to the people as most of the mobile devices and desktops users can manage to access these delivery mechanisms.

Another important focus of this work is to accommodate mobile users. Hence there is the need to explain mobility in existing systems and also in the proposed system because mobility is crucial for achieving the expected capability of sending notification to SANPARK subscribers' mobile devices and desktops.

2.6.2 Mobility in A Publish/Subscribe Architectural Framework.

The architectural model uses a mediator which receives notifications on behalf of a subscriber during disconnections. The mediator acts as a subscriber proxy, and can register interest in subscriber's location. When the subscriber reconnects to the system, the mediator will get a notification with the new subscriber's location, and then deliver the saved messages to the subscriber. An interesting part of the model is that it relies on the publish/subscribe infrastructure itself to transmit the information about changing subscriber locations. However, this raises a serious security concern. A malicious party could play the role of a mediator, track subscribers, therefore jeopardizing location privacy, and delivering bogus notifications after subscriber reconnection.

In chapter three a detailed account of the proposed model will be given to show how the mediator and implicit models were combined into the new proposed model. To the best

of the author's knowledge, all the existing systems that were represented in this work are based on models crafted using the already existing publish/subscribe models that were also represented in this thesis. Due to the understanding of the existing system in publish/subscribe communication paradigm and existing model of the publish/subscribe communication paradigm, the model that is further discussed in chapter three is crafted using some of the existing models of publish/subscribe paradigm to form one model as the solution to the problems given in this study.

CHAPTER THREE

MODEL DEVELOPMENT

3.1 Introduction

The increasing popularity of information services that rely on content delivery in mobile environments motivates the need for a mobile content dissemination service, which is an efficient and scalable service that enables the delivery of personalized and customized content to mobile and desktops users. Publish/subscribe middleware offers mechanisms for content personalization. Subscribers define the characteristics of content that is of interest to them in order to receive notification when such content becomes available. We list and analyze the requirements of the SANPARKS system and the content dissemination service supporting mobile users of SANPARKS business services. The possible design principles that can be handled by our model are analyzed and defined in this chapter. Our new model should accommodate both mobile devices and desktop users to ensure equal access to services.

This chapter is organised as follows: Section 3.2 states the requirements formulation of our system. Section 3.3 defines the design principles that drove the creation of the new model. Then the rest of the sections of this chapter define the development of the new model for SANPARKS system.

3.2 South African National Parks System Requirements

In order for SANPARKS customers to receive services information in a more concise manner than what is currently available. The organization needs to adopt the publish/subscribe communication paradigm. This will allow customers to subscribe to the system and set their preferences during the activation of the subscription in order to be notified with information that is relevant to their interest.

SANPARKS also need to consider the mobile commerce service model such that their business can be extended to mobile device users. This has a potential of increasing the number of customers that support the business as mobile devices are very popular and affordable. Mobile devices users will be able to enjoy the services information offered by SANPARKS anytime and anywhere.

It goes without saying that SANPARKS also need to give access to people that are using desktops. This category of people would like to gain access to information via SANPARKS desktop interfaces whether at home or in the office.

Therefore there is the need to restructure the current organisational architecture into an information-provider entity before the publish/subscribe communication paradigm can be adopted in the current system.

The wide acceptance of content information service depends on the precondition that the system delivers only highly personalized and customized content in accordance with user

preferences and current presence status. This may bring about the creation of a "branded" dissemination service that is invulnerable to spam. The service could become a trusted intermediary between content publishers and subscribers that filters information according to user's needs. The design requirements that need to be satisfied are outlined next.

Push-based content delivery: Service users must be able to define the type of content they want to receive, and be served with the published information as soon as it is available. The push-style content delivery eliminates the burden of querying for information at regular intervals and is in accordance with the stochastic nature of content creation and publication.

Content filtering and personalization: Content filtering is enabled through user subscriptions to minimize the number of received message that are not of interest. This feature requires that services are personalized and adopted to user context. The concomitant effect is that information overload on a user is reduce to the barest minimum

Scalability: This requirement connotes that service is optimized for the particular application area with respect to the number of publishers and subscribers in the system, and the size and frequency of published content.

3.3 Design Principles of the Publishing Personalized Data Architecture for SANPARKS

The Publish/Subscribe Architectural Framework for SANPARKS is based on two related models that were defined in section 2.6 which is the **broker (mediator) and implicit models** because they satisfy the following design criteria:

- Restructuring - Restructure the SANPARK system into an information provider;
- Transparent Information Delivery – Integrate a publish/subscribe engine that allows a consumer to get information without knowing the source and
- Personalization – Enhance the information delivery engine to serve mobile users in a personalized manner

The subsection to be discussed in the pages to follow detail the overview of the design principle that have been introduced earlier on.

3.3.1 Restructuring the organizational information Architectural Framework.

Some of the SANPARKS are interested in enabling most if not all the communities of South Africa to benefit from its activities. Historically, static web pages were used to inform the public about services or products offerings excluding the content that information consumers are willing to buy. If the current information delivery to customers through static web pages are not simplified and personalized then there will be no need to restructure the SANPARKS system into an information-provider entity that will make sure that information is simplified. Restructuring the existing entails identifying the system or component that would be modified to achieve the goal of simplifying information. The usability testing that will be done during testing phase of

this system would identify whether the new SANPARKS Information System developed is the information provider entity or not.

3.3.2 Making Information Delivery Transparent in a Publish/Subscribe Architectural Framework.

The SANPARKS Information architecture is currently not associated with any communication style and therefore we explore the obvious gap by introducing an delivery mechanism that allows customers to get information without knowing the source. Adopting the publish/subscribe communication mechanism provides no prior knowledge between the subscriber as the consumer and the publisher as the source, since the publish/subscribe communication paradigm is the engine that allows a consumer to get information without knowing the source. This design principle promotes the notion of making information sources transparent to the user.

3.3.3 Personalization in the Publish/Subscribe Architectural Framework

Personalization is about tailoring products and services to better fit the user, e.g. *by focusing on the user needs, preferences, interests, expertise, workloads, task etc.* The heart of this study is to bring user context into focus as the means of capturing all of the above. In this way, information delivery serve to both mobile and desktop users is done in a personalized manner. The notion of personalization in this work requires that the subscriber provides his/her preferences and profiles before the system can guarantee that the information published will meet the subscriber's expertise, needs and interest.

The foregoing design principles provide the basis for converting existing static web pages into a dynamic e-commerce based information delivery enterprise. In order to start trading conservation information as a commodity, via the Internet, the SANPARKS model had to be restructured as conceptualized in fig 3.1.

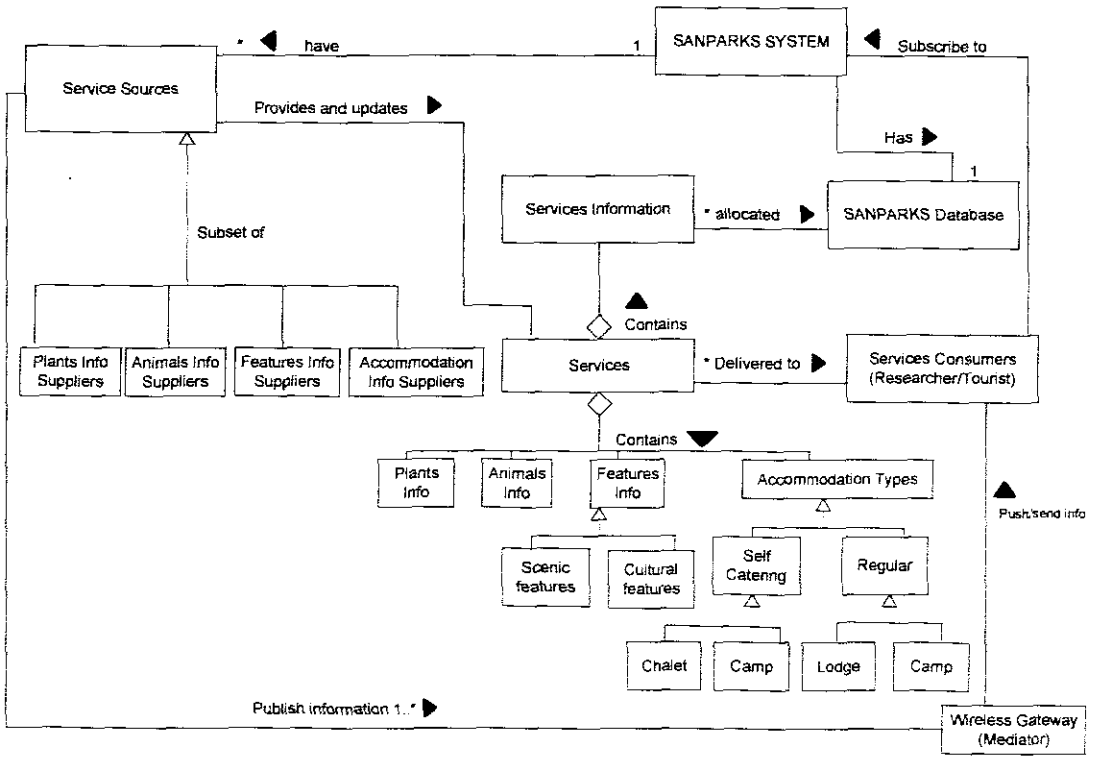

Figure 3. 1: The Restructured SANPARKS Conceptual Model

The SANPARKS conceptual model given in Fig 3.1 adopts a service-oriented approach to conceptually show information entities and their relationships. The goal is to enable the SANPARKS entities become information publishers. The overall information architecture of the restructured SANPARKS has been conceptually modeled. What remains is to describe the identifiable architectural elements.

- **SANPARKS SYSTEM** consists of the South African National Parks forming a unified business or system that provides service consumers with access to personalized information about services conserved in the SANPARKS, this business has many service suppliers.
- **Service Sources** are divided into four categories in the SANPARKS system; we have **Plants Info Suppliers, Animals Info Suppliers, Features Info Suppliers and Accommodation Info Suppliers.** In this case Info fully stands for Information. Service sources act as repositories for, and disseminate specific type of information commodity. A mediator exists through whom service suppliers publish their information to consumers.
- **Services** consist of the tasks performed in response to users requests concerning plants, animals, features and accommodation in most cases; information is retrieved, and packaged to meet users' specific needs.
- **Features information** can be categorized into scenic features and cultural features where scenic features can be mountains or a river, the cultural features can be a tribal village.
- **Accommodation types** can be classified into regular and self catering type of accommodation. Regular accommodation consists of the lodge type or camping type. The self-catering category consists of chalet or camping type as well.
- **Service Consumers** will subscribe in order to receive the available SANPARKS services information and products information matching their interest according to the preferences supplied at subscription time.

- **SANPARKS Database** holds the filtered services information of the distributed national parks that is ready to be disseminated to the relevant subscribers by the SANPARKS system using the mediator.
- **Mediator** acts as the software bus between the Service Suppliers and Service Consumers. (Eugster, et al., 2003) prescribe that a good publish/subscribe system should not allow Service Suppliers and Service Consumers to have prior knowledge of each other/or should not exchange services information directly. The mediator is the mechanism used to avoid direct communication between the two entities. The mediator waits for information to be published by Suppliers and matches published information to the relevant corresponding consumers, once the relevant consumers become available then the mediator pushes the published information to service consumers.

The conceptual model for the SANPARKS Information system already described above requires the newly constructed publish/subscribe architectural framework that support the implementation of adopting the publish/subscribe communication paradigm into the SANPARKS environment. This would enable the subscribers to receive the personalized services information using their desktops and mobile devices without knowing the source.

3.4 Publish/Subscribe Architectural Framework for SANPARKS System

The Architectural Framework presented in fig 3.2 combines both the mediator and the implicit models. This architecture seeks to support the newly conceptualized SANPARKS model in fig 3.1 in order to make it fit for the information provider entity that enables information consumers to receive the personalized information through their mobile devices or desktop computers. This architecture also presents the adoption of publish/subscribe paradigm into the SANPARKS environment.

The model ensures that communication between end points is anonymous, asynchronous and loosely coupled. In other words, the architecture ensures that the system decouples publishers and subscribers in time, space and flow. This decoupling of subscribers and publishers in time, space and flow makes publish/subscribe systems highly scalable by removing all explicit dependencies between the interacting parties. It also helps the system to adapt quickly to a dynamic environment. Decoupling in space allows the subscriber to move from one location to another without informing the publisher while decoupling in time allows for disconnected operations of the subscriber. The following section identifies and discusses all the component or building blocks used to craft the Publish/Subscribe Architectural Framework.

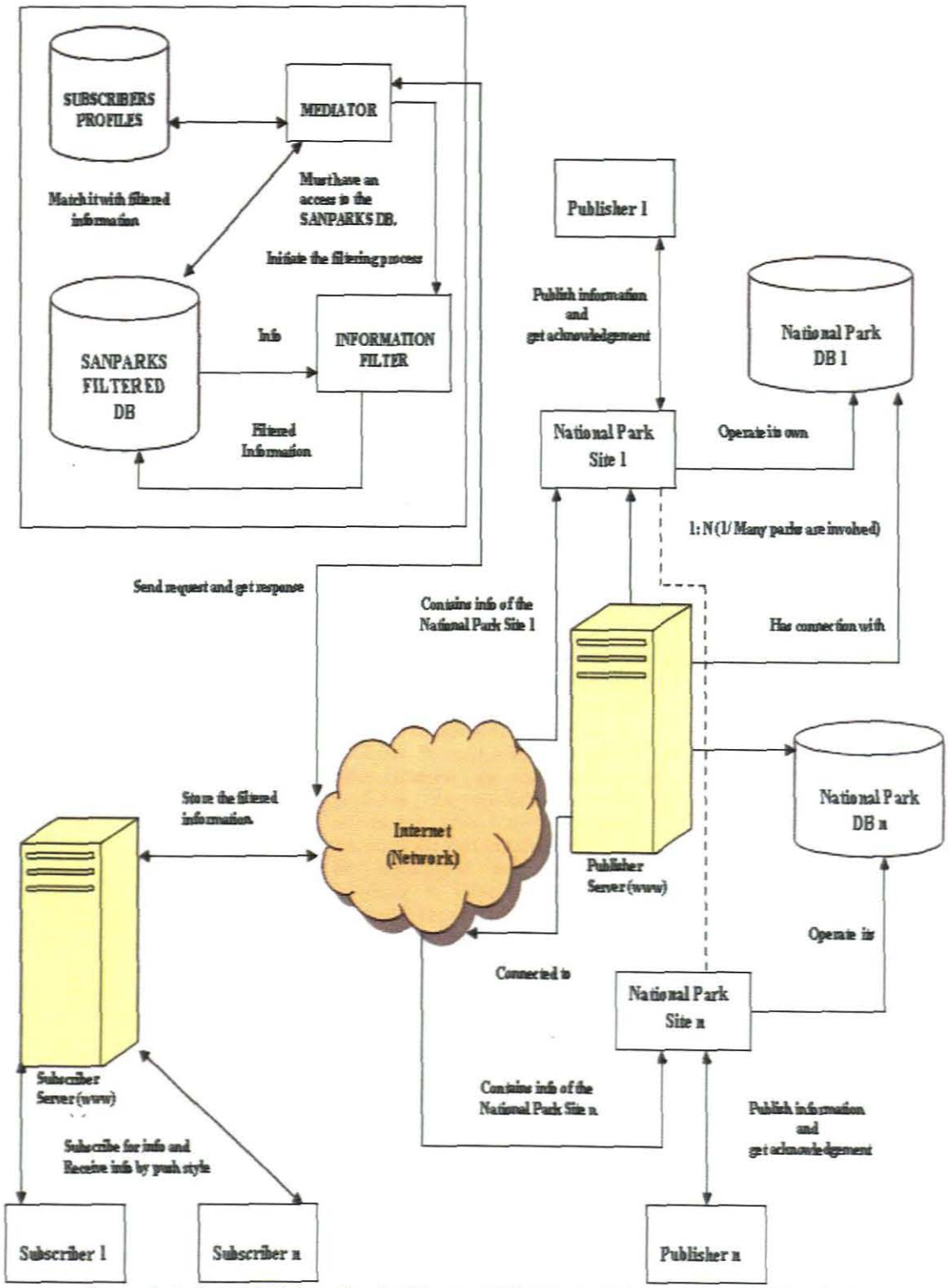

Figure 3. 2: Publish/Subscribe Architectural Framework for the SANPARKS System

3.4.1 Building Blocks and Components of Publish/Subscribe Architectural Framework for South African National Parks.

The architecture mainly comprises three different components together with their nodes as the buildings blocks of the entire architecture. The components and nodes of the architecture are defined using the tabular form as follows:

Table 3. 1: The entire components and building blocks of the Architecture

Publisher Component	Mediator Component	Subscriber Component
Publishers: This node declares its intention to send publications and describe publication types and patterns. Generally this node publishes all the information that is available about SANPARKS services to the mediator. **Publisher's Server:** This server is connected to the network and it holds all information of the national parks databases.	**Mediator:** This node is the software bus which allows the communication between the publishers and subscribers. It is also known as the controller since it is responsible for controlling the entire functionality of the architecture. This node allows the exchange of information between the publishers and subscribers. This node ensures the transparent information delivery between the subscribers and publishers. **Information Filter**: This node is the one that summarizes and categorize the services information send by the publisher. We call it the information filter because as the publisher publishes the available services information to the mediator automatically that information has to be summarized and categorized according to topics/preferences by this node. The mediator interacts with this node by alerting it to filter the distributed information before it can be stored on the SANPARKS database. **User Profiles:** This node holds all the information of the subscribers which is needed by the mediator during the matching phase in order to determine the relevant subscribers to receive the information that is send by the publisher to the mediator. **SANPARKS Filtered Database:** This database holds the filtered services information of the distributed national parks that is ready to be disseminated to the relevant subscribers by the SANPARKS system.	**Subscribers:** This node registers interest in receiving publications and specifies the subscriptions to the subscribers' server. This node waits for the information to be pushed to him/her in order to gain access to it by viewing that information. **Subscriber's Server:** This server holds all the information requested by the network from the mediator in order to push it to the relevant subscribers when they reconnect to the network.

3.4.2 Functional Requirement of the System

3.4.2.1 Use Case Diagram

This section presents the use cases for the prototype of the publish/subscribe architectural framework for SANPARKS to be implemented in this research work for usage in the SANPARKS organization.

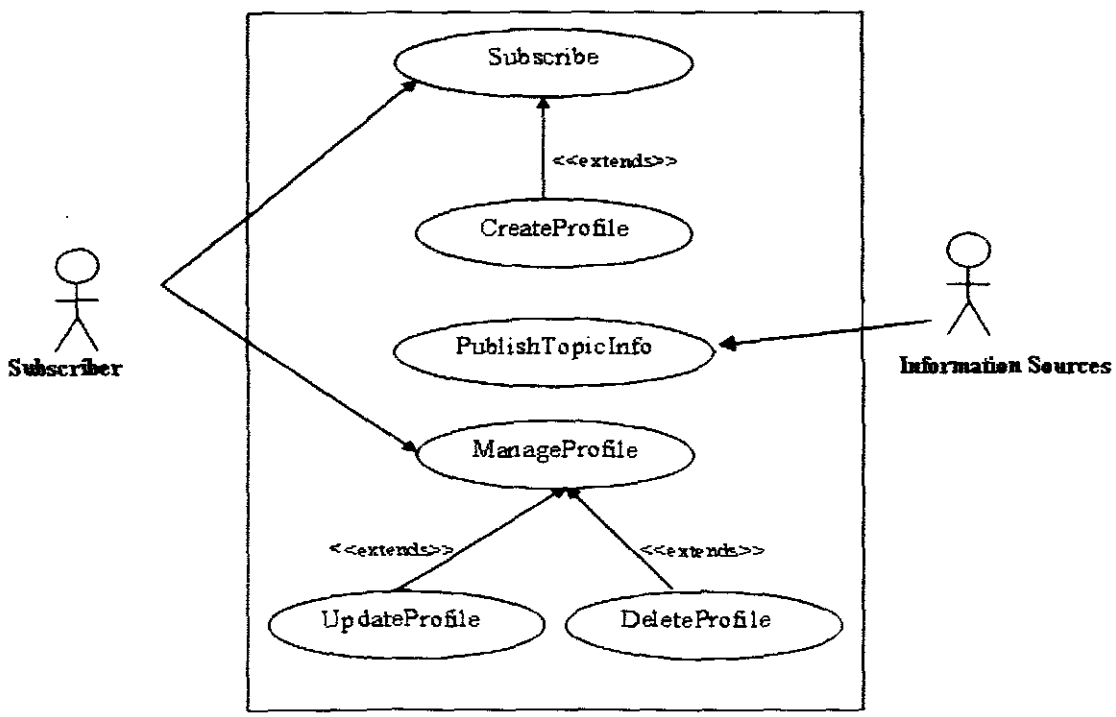

Figure 3. 3: SANPARKS Use Case Diagram

The usage scenarios of the system indicate that the system has two actors that are participating in the system as the publisher and the subscriber. These actors determine the occurrence of the adopted publish/subscribe communication pattern in the SANPARKS domain. In detail, a **publisher** in this system is the information sources of plants, animals, features and accommodation. The information sources deal with publishing SANPARKS services information as the house-keeping process rather than a public process to the mediator in order for the mediator to notify the subscribers with the SANPARKS services

information that meets their interest. The **subscriber** can subscribe for the topics of his/her own choice on the mediator in order to be notified with the SANPARKS services information which meets his/her preferences that s/he specifies during the subscription process on the system. The overall functionality of the system use cases is given below.

Subscribe: A subscriber can subscribe for the topics on the mediator to be able to receive notification about the available SANPARKS services information of his/her interests.

CreateProfile: A user defines the profile when s/he first uses the system for the system to authenticate him/her when s/he uses the system in future. The user needs to specify the personal details for the creation of the profile. The information to be provided by the user is the username, password, full name, e-mail address, and mobile phone number.

PublishTopicInformation: The information sources of plants, animals, features and accommodation publishes the SANPARKS services information to the mediator, and then the information is filtered by the mediator's information filter according to the appropriate topics category that is stored on the SANPARKS database. Finally the mediator as the controller of the system matches the services information that is on the SANPARKS database with the user profiles of the system and disseminates that information to the corresponding subscribers.

ManageProfile: The subscriber is able to manage his/her profiles from the SANPARKS Information System. Managing Profiles can be achieved by updating or deleting. Updating the profile occurs when the user makes some changes in his/her profile.

Deleting the profile occurs when the user prompts the system to permanently withdraw or delete his/her profile from the system database.

UpdateProfile: This use case is part of managing the profile by the system users. This indicates that the subscriber of the system can be able to update his/her profiles.

DeleteProfile: This use case is also part of managing the profile by the system users. This indicates that the subscriber of the system can be able to update his/her profiles.

3.4.2.2 Sequence Diagrams

Based on the use cases identified above, the following sequence diagrams were constructed in order to show the flow of information of the above use cases:

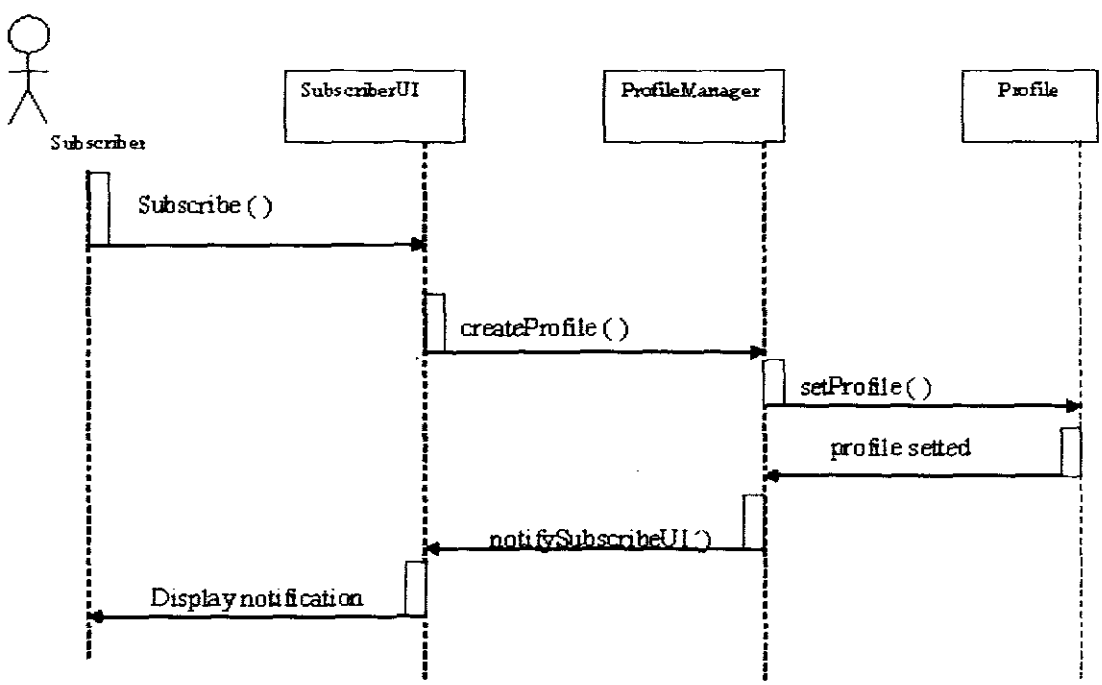

Figure 3. 4: Sequence Diagram for Subscribe Use Case

Figure 3.4 shows a sequence diagram of the Subscribe use case. The flow of time is shown from top to bottom, that is, messages higher on the diagram happen before those

lower down. The arrows (links) are messages - operation calls and returns from operations. In the SANPARKS domain the subscriber can be either a researcher or tourists who use the system for the first time. The subscriber uses the graphical user interface provided in the national park system to define his/her profile. In the process of defining the user profile, the subscriber provides his personal details (id, cell number, e-mail address etc) and preferences (plants, animals, features and accommodation as the additional feature) for the system to be able to identify and authenticate him/her when s/he access the SANPARKS services. For mobile phone users, the system should use the mobile phone number as the username and allow the subscriber to create his/her own password. The subscription and provision of preferences makes it easier for the system to present the user with relevant information during the push-based dissemination of information to avoid time consuming if accessing to the information. The following Fig 3.5 is the PublishTopicInfo sequence diagram that is explained in details below the figure.

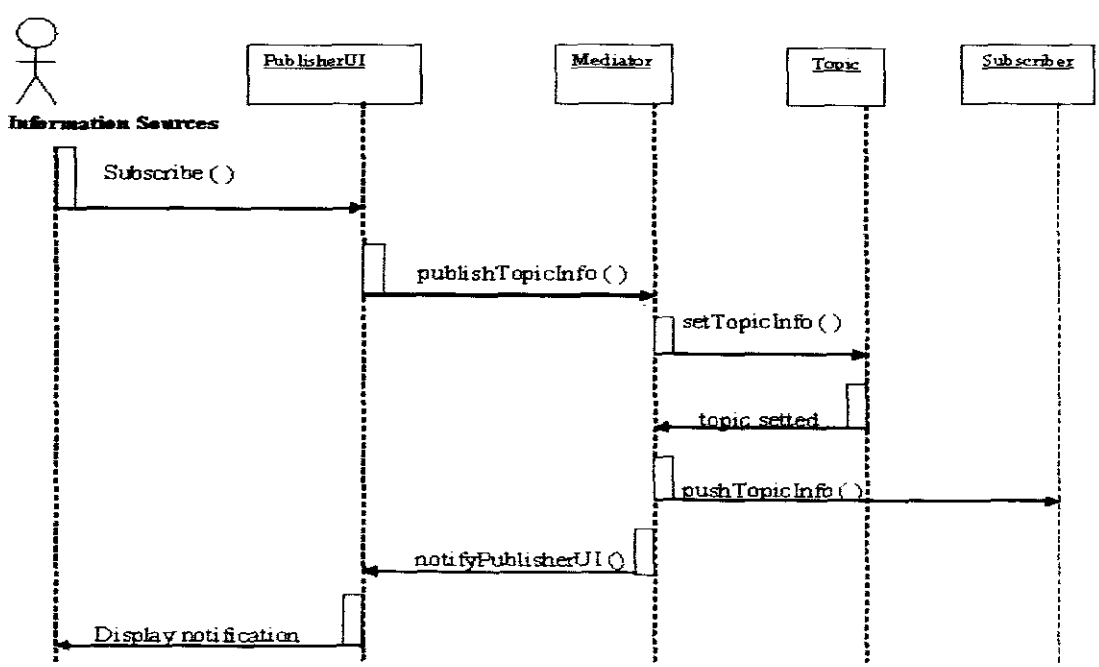

Figure 3. 5: Sequence Diagram for PublishTopicInfo Use Case.

Figure 3.5 shows a sequence diagram of a PublishTopicInfo use case. The flow of time is shown from top to bottom, that is, messages higher on the diagram happen before those lower down. The arrows (links) are messages - operation calls and returns from operations. In the SANPARKS domain the information sources are responsible for publishing information which is done as house-keeping process rather than public process. The information sources use the graphical user interface provided behind the scene to publish the information to the mediator. The reason of publishing information behind the scene is that SANPARKS organization is not interested in observing the way of publishing information to the subscribers but to ensure that subscribers receive the simplified and personalized information at the end of the day. The mediator then sets the topics information according to the relevant subscribers and pushes that information to the relevant subscribers.

Finally the mediator notifies the publisher as to whether the information was sent successfully or not. The figure 3.6 is the ManageProfile sequence diagram that gives the overview of the flow of information in the ManageProfile use case. The flow of information takes place when the subscriber activates the manage profile process in order to update or delete his/her profile. The subscriber interacts with the manage profile graphical user interface to either delete or update their profile. In updating the profile, the subscriber needs to provide the system with the new information to replace the one h/she is updating. In deleting the profile the subscriber need not provide the system with any information; all h/she needs is to specify that h/she is deleting his/her profiles on the system. The profile of the subscriber is then permanently deleted by the system.

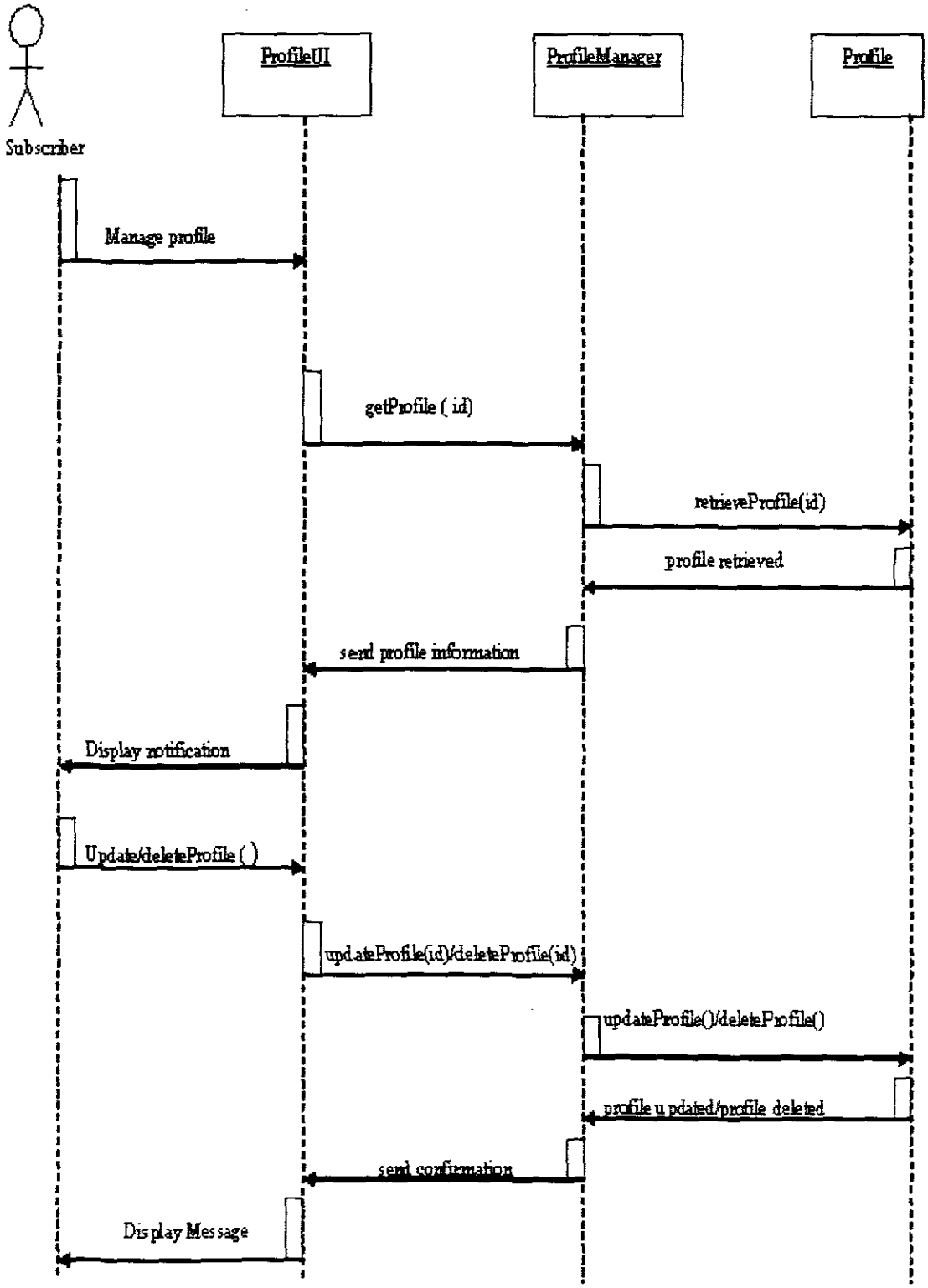

Figure 3. 6: Sequence Diagram for ManageProfile Use Case

3.4.2.3 Algorithms

This gives the overview of the use case specification using program design language that shows how the system use cases are going to be functioning after the completion of the system.

```
Use Case Name: Subscribe
Begin:
      1. request to subscribe
      2. enter subscriber information
      do
          check information completeness
              if(information is incomplete) then
                    display error message
                    goto step 2
              else
                    check information validity
                    if(information is invalid) then
                          display error message
                          goto step 2
                    else
                          persist information to profiles
                          notify subscriber
                    end if
              end if
          until(subscriber information is successfully persisted)

      end until
End
```

Figure 3. 7: Subscribe Algorithm

The figure 3.7 shows the algorithm of the Subscribe use case that gives an overview of the functionality of the subscribe use case. The subscriber requests to subscribe to the system then the system provide the subscriber with the subscribing user interface for the subscriber to fill in the required information. On completion of providing the system with the required information, the system checks the validity of the information provided by the subscriber. If the information is valid the system successfully registers the subscriber, but if the information is invalid the subscriber is then requested to re-enter the valid information.

```
Use case Name: Manage Profile
Begin:
   1. request to manage
   2. enter credentials
   3. verify credentials

   if(credentials are invalid)
           display error message
           goto step 2
   else
                   5. choose either to delete or update

                   if (update is chosen)
                           display existing information
                           6.enter new information
                           do
                                   7.check information completeness
                                   if(information is incomplete) then
                                           display error message
                                           goto step 6
                                   else
                                           8. check information validity
                                           if(information is invalid) then
                                   display error message
                                   goto step 6
                           else
                                   persist information to profiles
                                   notify subscriber
                           end if
                   end if
                   until(update process is successfully completed)

                   else if(delete is chosen)
                           do
                                   9. delete record using an id
                                   if(delete is successful)
                                           notify user
                                   else

                                           display error message
                                           goto step 9
                                   end if
                           until(delete process is successfully
   completed)
                   else
                           //Do nothing
                   end if


   end if

   End
```

Figure 3. 8: Manage Profile Algorithm

Figure 3.8 is for the Manage Profile algorithm that gives an overview of the functionality of the ManageProfile use case. The subscriber as an actor in the SANPARKS model is responsible of managing his/he profiles in terms of deleting or updating the profiles. The subscriber can also update his/her profile by editing the information which is no longer relevant to him/her and provides the system with the new relevant information.

```
Use Case Name: PublishTopicInfo

Begin:
      1. request to publish
      2. enter credentials
      3. verify credentials
      if(credentials are invalid)
            display error message
            goto step 2
      else
            enter topic information
                  do
                        check topic information completeness
                        if(topic information is incomplete) then
                              display error message
                              goto step 2
                  else
                        check topic information validity
                        if(topic information is invalid) then
                              display error message
                              goto step 2
                        else
                              persist topic information to a database
                              notify publisher
                        end if
                  end if
                  until(topic information is successfully persisted)
      end until

   end if
End
```

Figure 3. 9: Publish Topic Information Algorithm

The figure 3.9 is the Publish Topic Information algorithm that gives an overview functionality of the PublishTopicInfo use case. The information sources are responsible of performing this process by interacting with the user interface that is provided behind

the scene of the SANPARKS Information System because this process is meant for house-keeping process rather than the public process.

3.4.2.4 Activity Diagram for the SANPARKS System

Figure 3.10 is a UML activity diagram showing actions taken during a session of accessing the service by the subscriber in the SANPARKS Information System. The initial action taken in order to be successful in accessing the service is to subscribe to the system. This will allow the subscriber to read message that has been sent to the desktop as e-mails or to the subscriber mobile device as a sms. Subscribing to the system also allows subscribers to update or delete profiles. The actions of updating and deleting profiles are only possible if subscribers have been authenticated by the system as the owners of the profiles they need to delete or update. All the actions discussed in this section can be achieved by using the desktop and mobile device for only subscribing to the system.

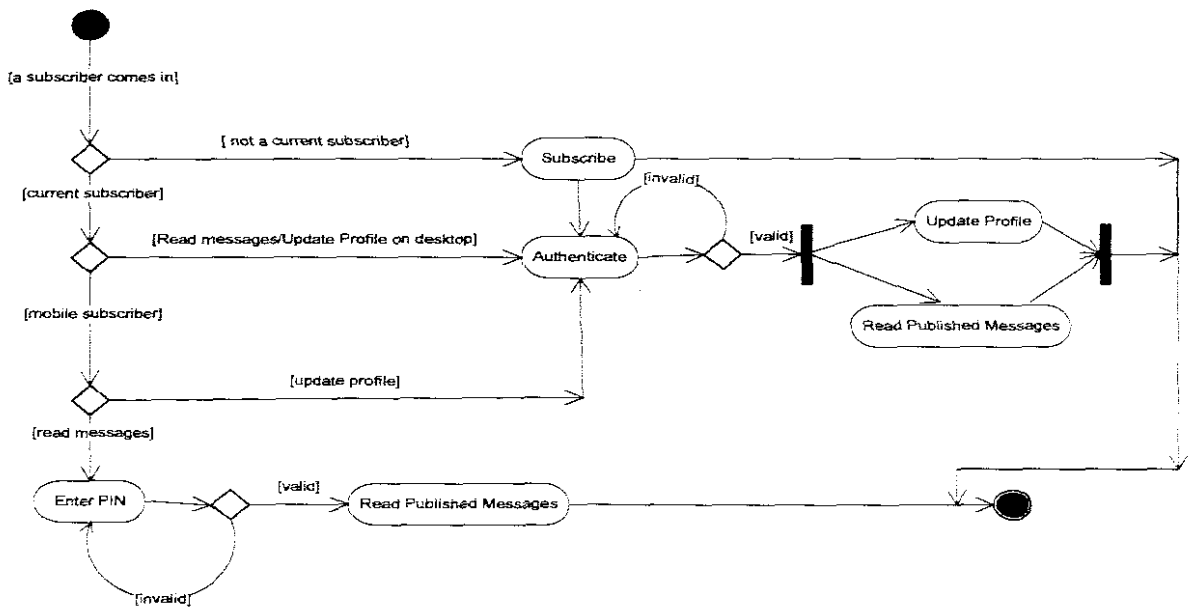

Figure 3. 10: SANPARKS System Activity Diagram

3.4.2.5 Class Diagram

Figure 3.11 is the class diagram that represents the classes of the system implementation together with their attributes and operations (methods). The implementation of the system is covered by nine classes as follows:

Publisher: This class represents a bean that defines properties and behaviors for handling information of the publisher entity.

Subscriber: This class represents a bean that defines properties and behaviors for handling information of the subscriber entity.

Message: This class is responsible for storing messages in an XML storage format which acts as a mailbox for emails. The XML storage is only for subscribers who chose email as a delivery method. The Message class defines methods for saving, deleting and retrieving emails. Furthermore the class defines methods for marking read messages.

TopicObserver: This class is responsible for listening to messages published to a topic, and then sends the message to potential subscribers when publication is received. This class applies the Observer design pattern in modeling collaboration between system objects.

Topic: This class represents a bean that defines properties and behaviors for handling topic publications. This class is a convergence of four properties the title, category, body and a keyword.

Profile: This class represents a generic class for both the subscriber's profile and the publisher's profile. Furthermore, it is a bean class with a defined properties and behaviors that are supposed to be shared among its descendant.

SubscriberProfile: This class is a derivation from the super class Profile. Added to this, it is a specialized bean that defines properties and behaviors for handling information pertinent subscriber profile.

PublisherProfile: This class is a derivation of the super class Profile. It is also a specialized bean that defines properties and behaviors for handling information pertinent publishers' profile.

Preferences: This is a delegate in class SubscriberProfile and it is responsible for capturing and querying subscriber preferences. It defines attributes that enables subscriber preferences to be persisted to permanent storage.

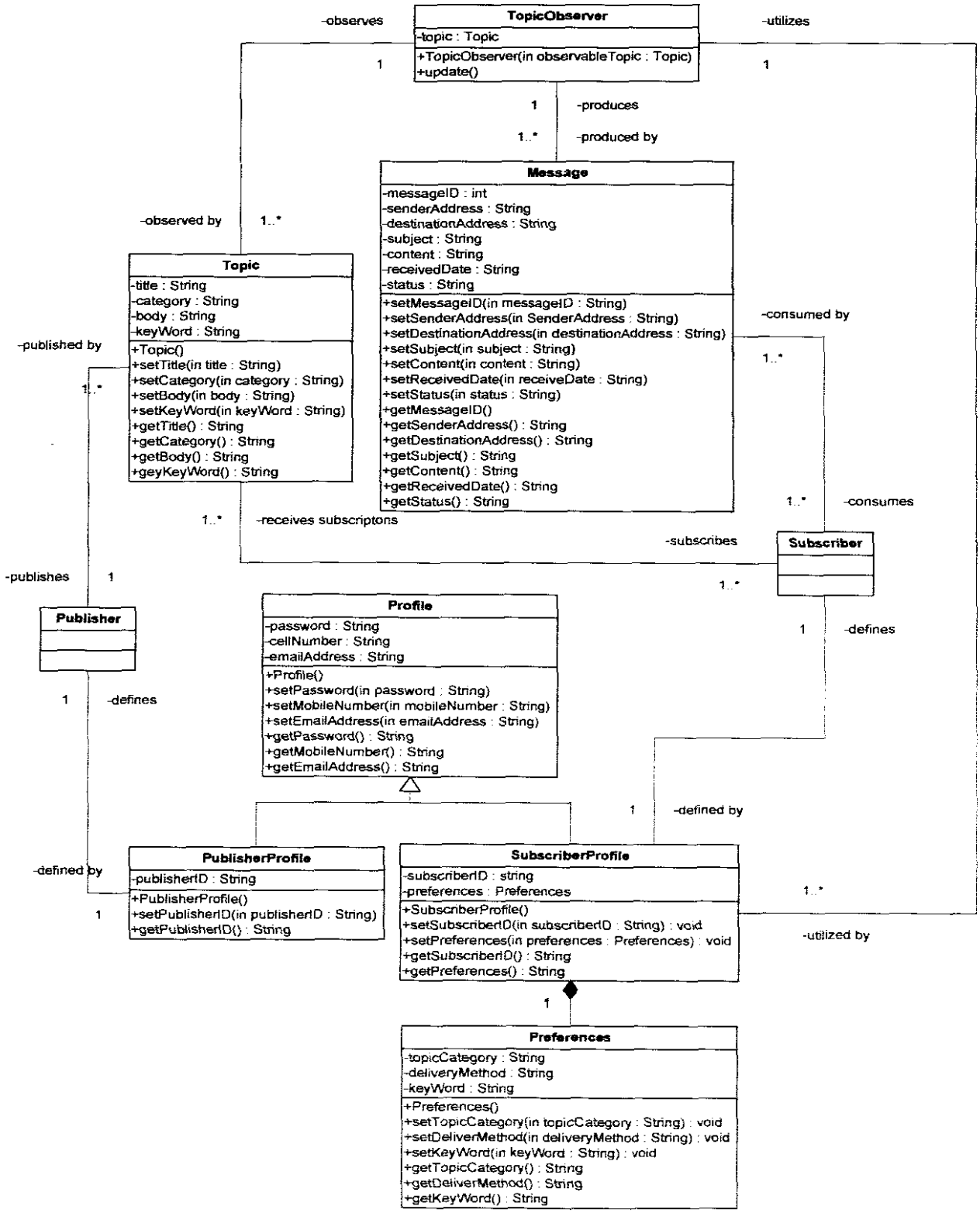

Figure 3. 11: The Publish/Subscribe SANPARKS Class Diagram

3.4.2.6 Deployment Packages

Fig 3.12 illustrates a business package for handling subscribers' requests for subscribing to the service. This package illustrates a business model that defines business classes which models the functionalities of subscription. The business model illustrated in the above subscription business package is comprised of business classes that can be implemented as bean components, such as classes like Message, Profile, Preferences and SubscriberProfile. Only the SubscriberProfile class attributes can be persisted to a permanent storage.

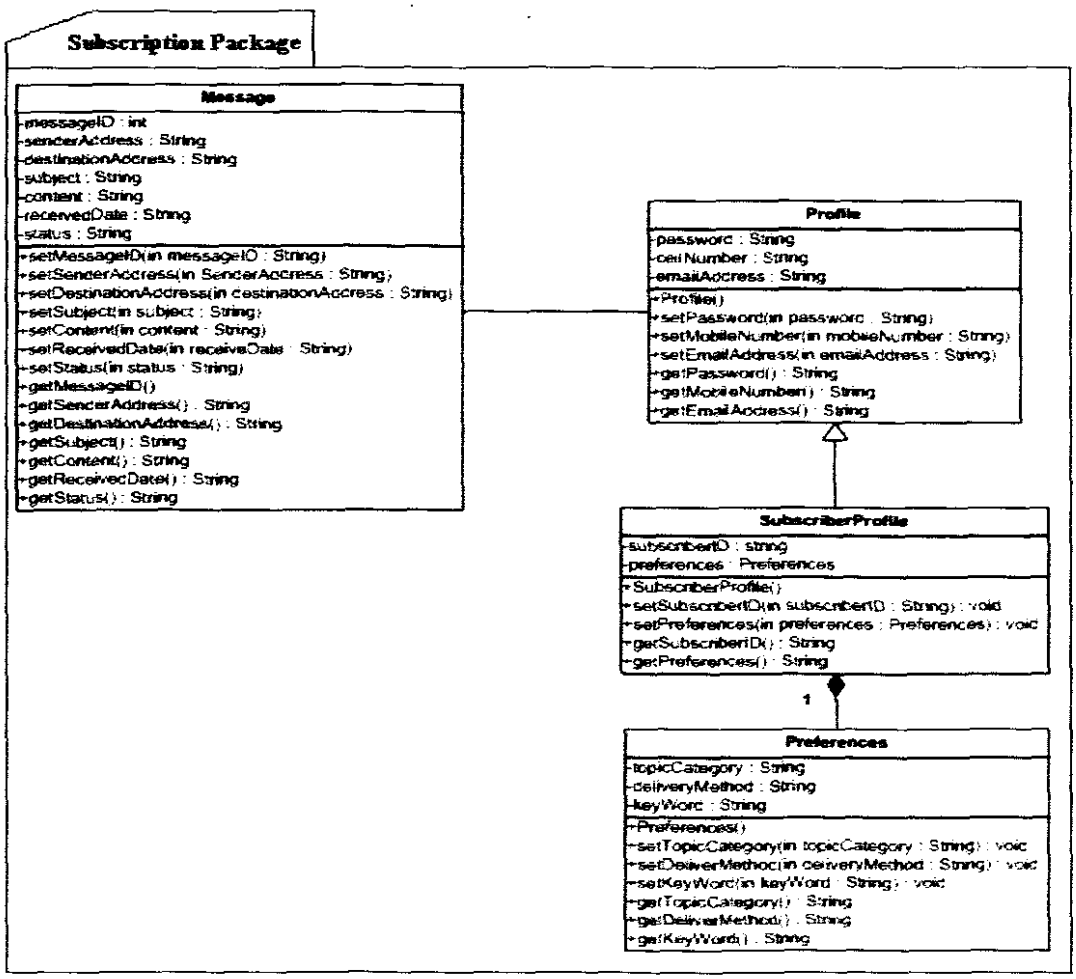

Figure 3. 12: The Publish/Subscribe Subscription Package

Fig 3.13 is a Publish/Subscribe publishing package and defines a business model responsible for handling the publishing request initiated by the publisher. The business model defines business classes, which can be implemented as bean components such as classes like Profile, PublisherProfile and Topic. Both the Publisher and Topic class attributes can be persisted to a permanent storage.

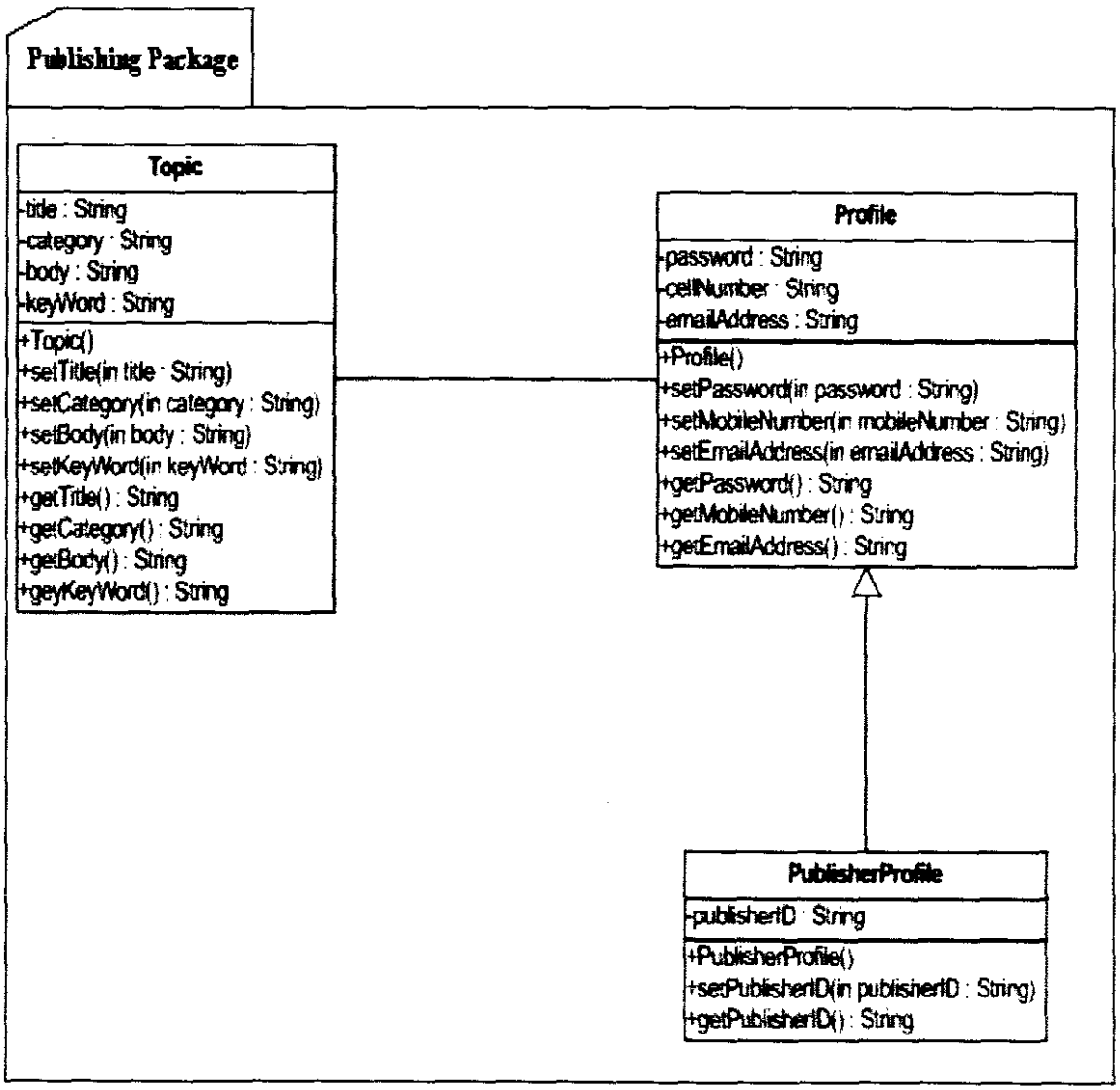

Figure 3. 13: Publish/Subscribe Publishing Package

3.4.2.7 Access Layer

Fig. 3.14 depicts the entire access layer package the SANPARKS architecture. This package implements access methods responsible for accessing data stored in the XML database.

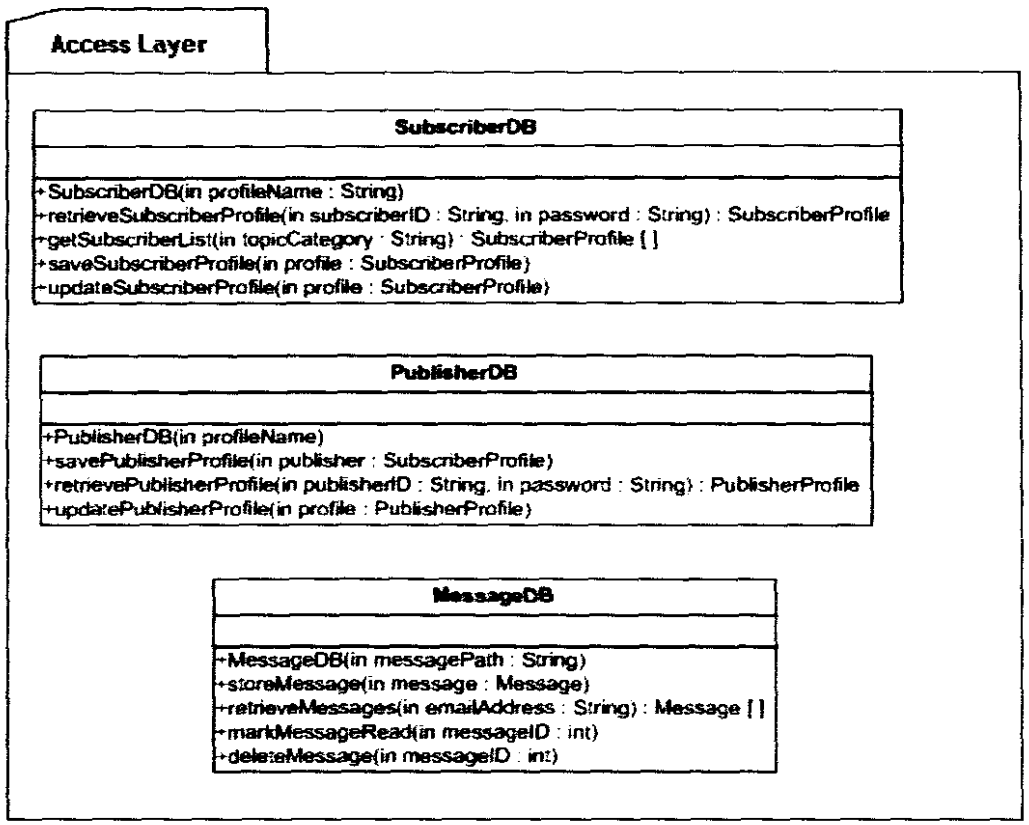

Figure 3. 14: Access Layer of the SANPARKS Information System.

The Access layer package is comprised of three core access classes namely the SubscriberDB, PublisherDB and MessageDB.The SubscricerDB defines access methods responsible for accessing subscriber profiles in an XML database, and the PublisherDB does the same as SubscriberDB.Lastly the MessageDB defines access methods responsible for storing, retrieving and marking emails in an XML store.

CHAPTER FOUR

IMPLEMENTATION AND EVALUATION OF THE PROTOTYPE

4.1 Introduction

The previous chapter presented the overall development of the proposed architectural model for publishing personalized data in the access and provision of both e-services and m-services. The focus of this work is on providing personalization of services within the SANPARKS domain for the information consumers to gain access to information that meets their interest. This chapter presents the design, implementation and evaluation of the proposed publishing personalized data model discussed in Chapter 3.

4.1.1 Description of the Implementation

The SANPARKS architecture is implemented as a web application that supports publication of messages through simple web forms processed by simple servlets running in a web container. The published messages are disseminated to potential subscribers either through email or SMS, dependent upon which delivery method the subscriber specified during subscription. The support for varying message dissemination heavily relies on the architecture's focus on both mobile and web subscribers. Emails are stored as simple XML documents accessed using the JAXP (Java API for XML Processing) and SMSs are sent to potential subscribers using the WMABridge API. The WMA Bridge API enables J2SE (Java 2 Standard Edition) applications to easily interface with MIDlets defined by the J2ME specification through messaging.

The SANPARKS architecture is crafted as a three tier implementation architecture which is comprised of the client, middle and the information tier. The client tier is a convergence of two MIDlets that are developed under J2ME specification and tested using the Sun Wireless Toolkit 2.3 Beta version. The first MIDlet defines a form that enables users to subscribe to the content dissemination application. The second MIDlet defines an interface for receiving incoming messages published by SANPARKS legitimate publishers. The middle tier is a convergence of various packages structured into three standard components of business logic, access logic and presentation logic. The middle tier runs under a web container basically the servlet container which is Apache Tomcat Web Server 5.0.The web container is a module that handles processing of web components, managing various aspects such as state management, concurrency control, thereby giving the developers the freedom of developing an application without worries of system dependent mapping and calls. The information tier is a collection of XML documents which is referred to as an XML database. The collection is accessed through JAXP for reading and writing to XML documents. The implementation adopted the use of DOM Parser for it's capable of structuring information into a hierarchical tree.

4.1.2 Environment Specification

The implementation of the software was carried out on Borland JBuilder 2005 Enterprise Edition and it was tested using JBoss Application Server 4.0.5.The mobile environment was simulated using the Sun Wireless Toolkit 2.3 Beta version J2ME Emulator from Sun Microsystems. J2ME Emulator was configured to support one of the MIDP optional API, the Wireless Messaging API (WMA) to facilitate message communication in the mobile

environment during the communication with the server. The client application is developed to run on J2ME devices that support CLDC 1.0 and MIDP 1.0 and also support communication through HTTP networking.

The implementation of the server was accomplished using one of the J2EE compliant servers, the JBoss Application Server 4.0.5.The server-side components are configured to run on any server that conforms to the J2EE Specification. The server handles messages from clients through the use of WMA Bridge API, which facilitates message communication on the server-side during a session with the client. The server is comprised of web components that run within the Servlet container and also uses the JAXP to communicate with external data source defined outside its boundary. The data source is the information source of the entire SANPARKS application and it is defined using an XML database, a collection of xml files stored within the same directory with the application.

The application was tested on a desktop machine running Windows XP Professional Edition as an operating system. The machine was equipped with an Intel Pentium IV processor with a processing speed of 3 GHz and 512 MB of RAM. The application consumed 5.5 MB of hard-disk storage.

4.1.3 Implementation Model

Fig. 4.1 illustrates a Publish/Subscribe implementation model which comprises three tiers, namely the client tier, middleware tier and information tier. The client tier can either be J2ME or web clients communicating with web components running within the web container in the J2EE Server (the middle tier). The middle tier is a convergence of web components running in the web container (Servlet engine) and business components within the application logic. Lastly the information tier is implemented in terms of an XML database which is a collection of XML (Extensible Mark-up Language) files securely persisted on a permanent storage. The interface between the application and the XML database is defined using JAXP (Java API for XML Processing).

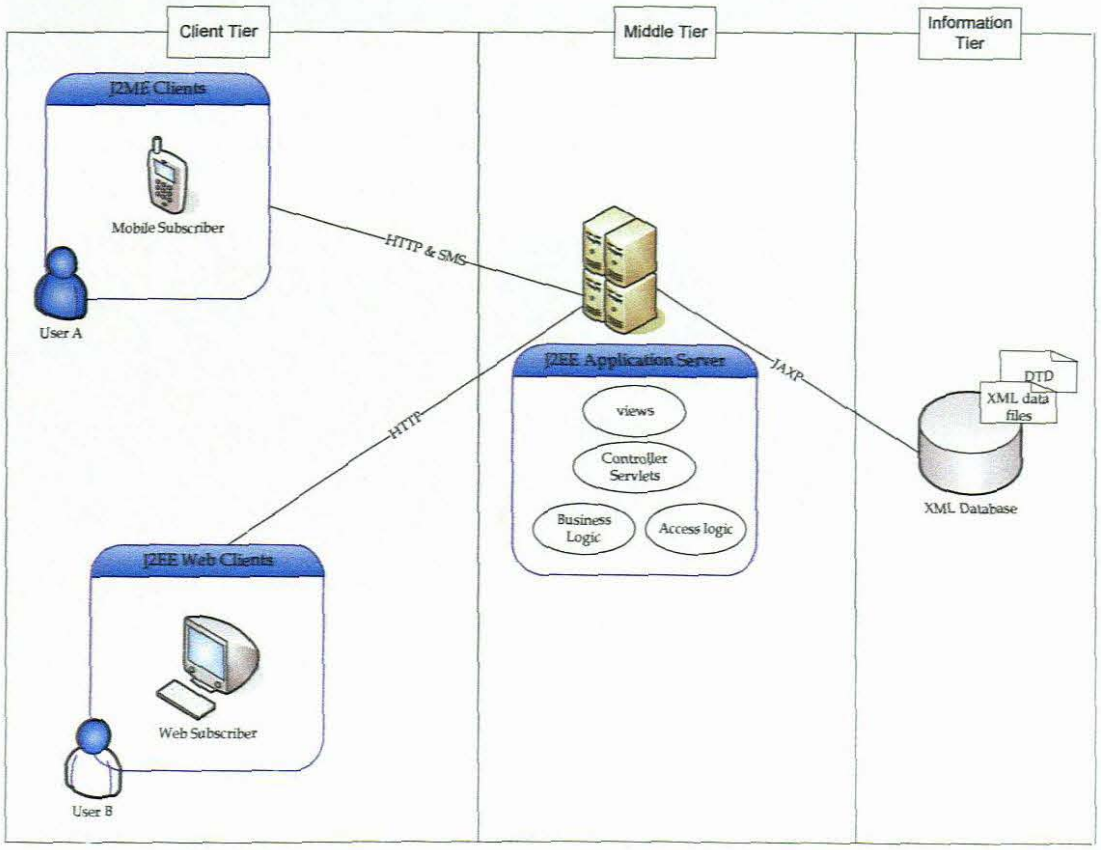

Figure 4. 1: The System Implementation Model

4.2 Implementation Screenshots

This section presents some of the interfaces used to fulfill the adoption of the publish/subscribe communication paradigm into the SANPARKS Information System.

4.2.1 Subscribing and receiving e-mails on the desktop.

Figure 4.2 is the desktop portal interface that information consumers use to subscribe to the SANPARKS organization, to view the services information delivered as e-mails and just to take a tour on the new SANPARKS Information System.

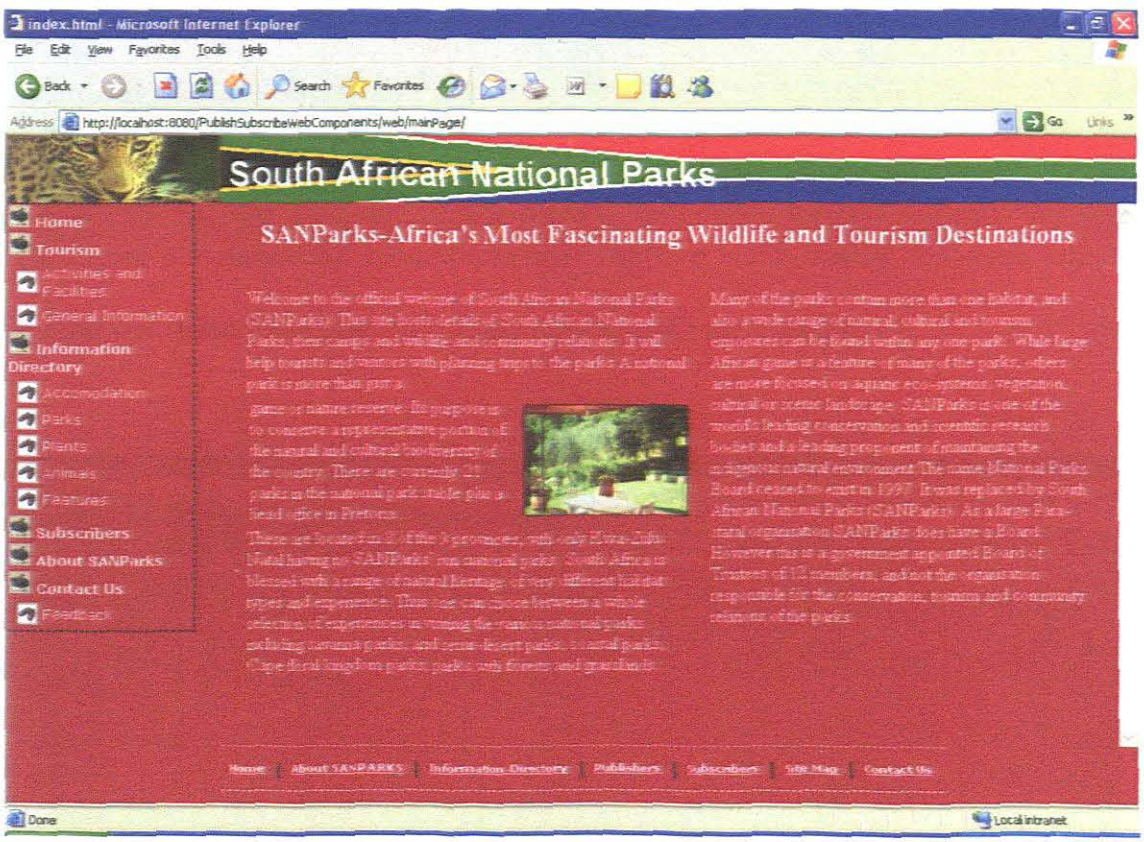

Figure 4. 2: A portal interface for SANPARKS information consumers.

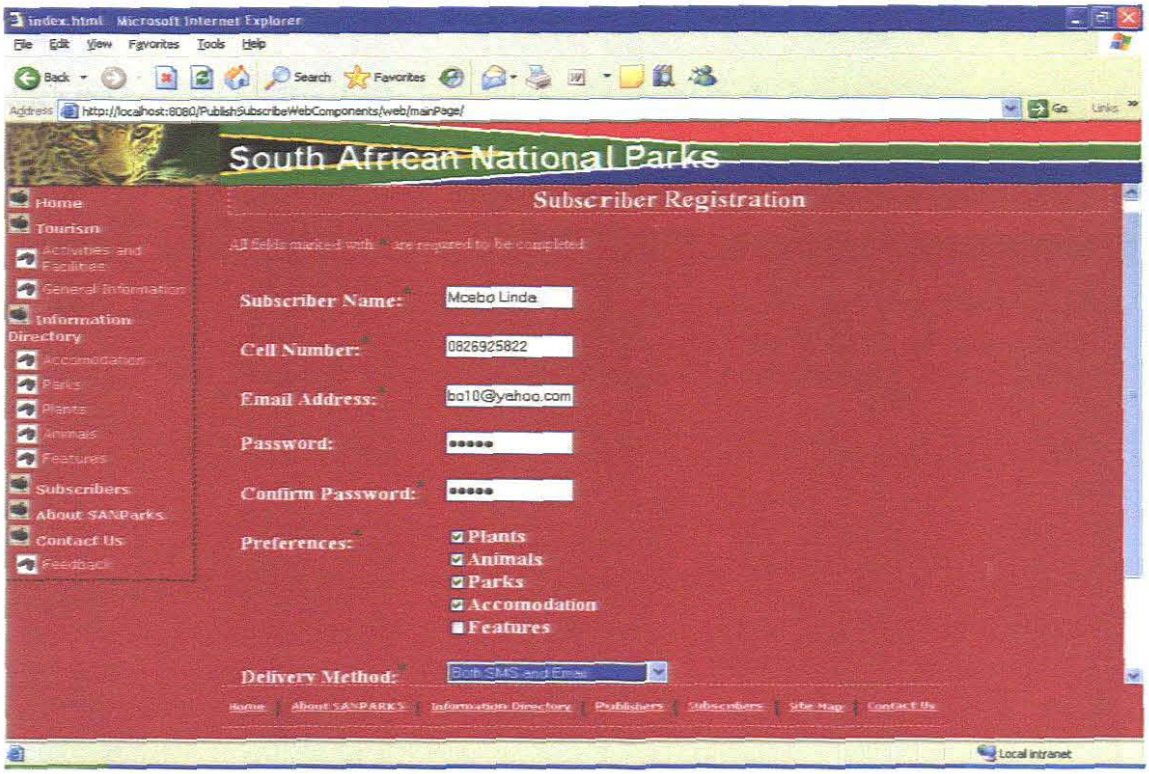

Figure 4. 3: Subscriber's Registration Interface on the Portal.

The figure 4.3 shows the subscriber's user interface which assists first time users to register and subscribing users to indicate their specific preferences.

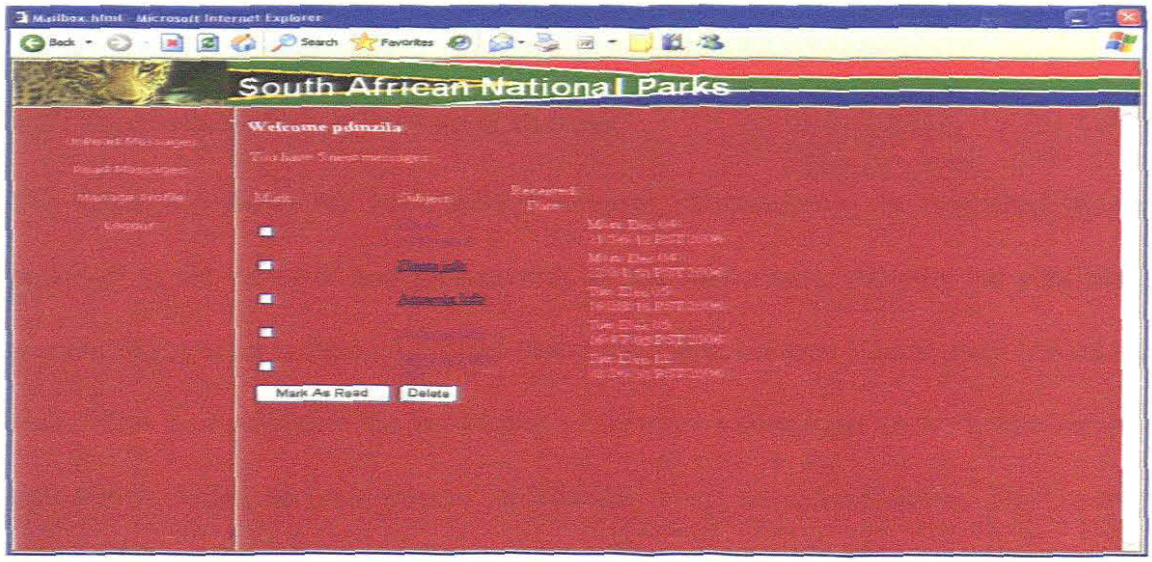

Figure 4. 4: The list of E-mails on the Desktops.

Figure 4.4 shows a list of sent e-mails as notification on the desktop. The user needs to click the email of his/her choice in order to gain access to the services information.

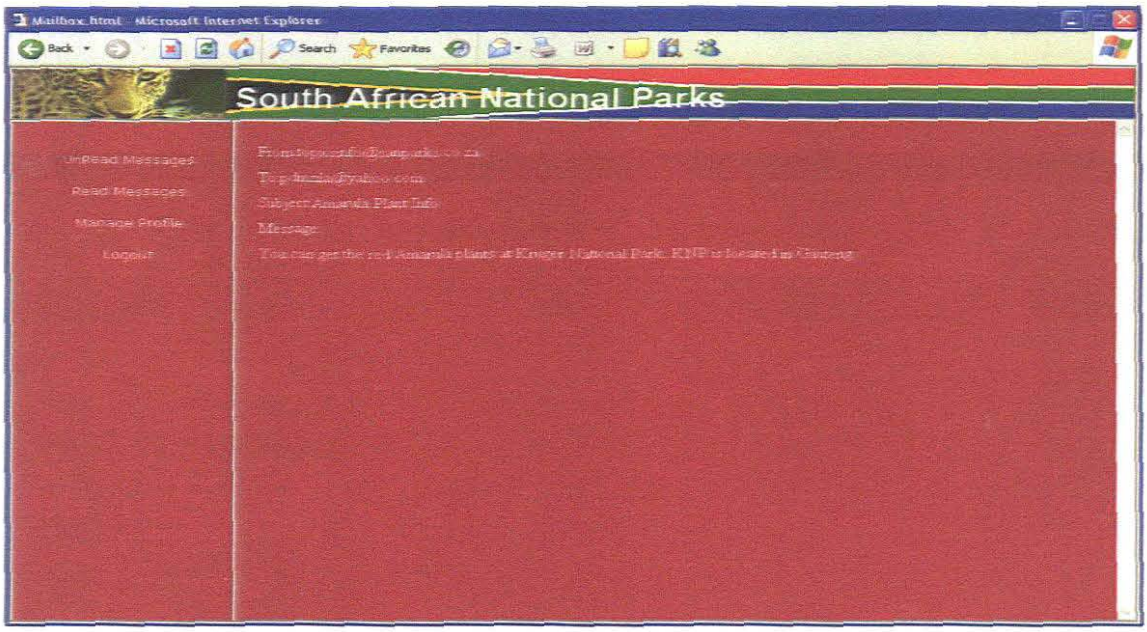

Figure 4. 5: The service information delivered as an e-mail.

Figure 4.5 shows a notification delivered as an e-mail in response to a subscription. The information consumer has the option of marking this e-mail as a read message by sending it to the read message folder or deleting this e-mail.

4.2.2 Service Information Publishing Process.

The figure 4.6 shows the SANPARKS services information to be published by the publisher to the mediator and subsequently to the subscribers. Anyway the publishing process taking place in figure 4.6 is not meant for public because it takes place behind the scene. This research takes the publishing of services information as the house-keeping task that is meant for maintenance behind the scene instead of public. This is only because our research focused on enabling the subscribers to receive the simplified and

personalized information, rather to observe the way of publishing the services information by the information sources. The figure 4.6 is not meant for the public, the reason why it is shown here is to enable the reader to see that the information received by the subscribers is published instead of coming from nowhere.

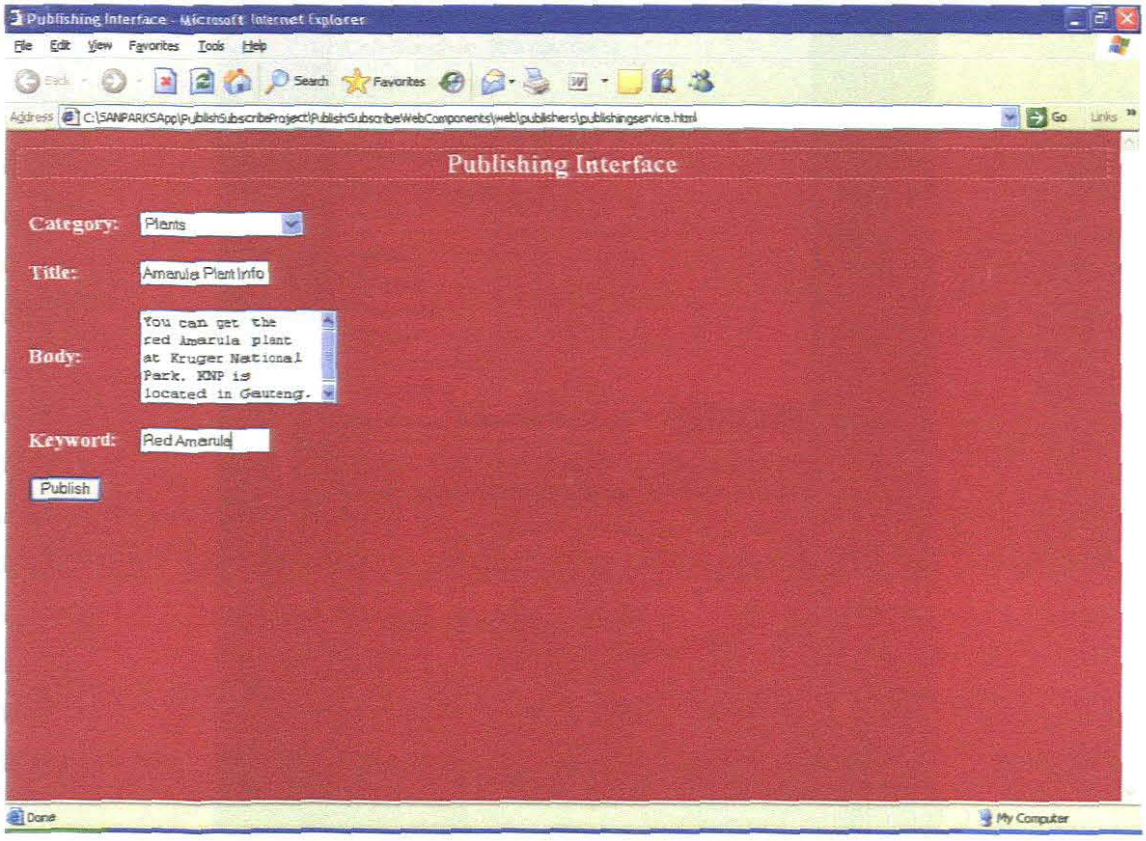

Figure 4. 6: Publishing service information using the desktop.

4.2.3 Service Subscription using a mobile device.

Figure 4.7 shows the mobile user interface subscribers use to subscribe for the SANPARKS services information. The user subscribes by giving his/her details which are limited to name, mobile phone number, email address and a password. Then, one or more preferred topic(s) must be specified before the subscription is submitted.

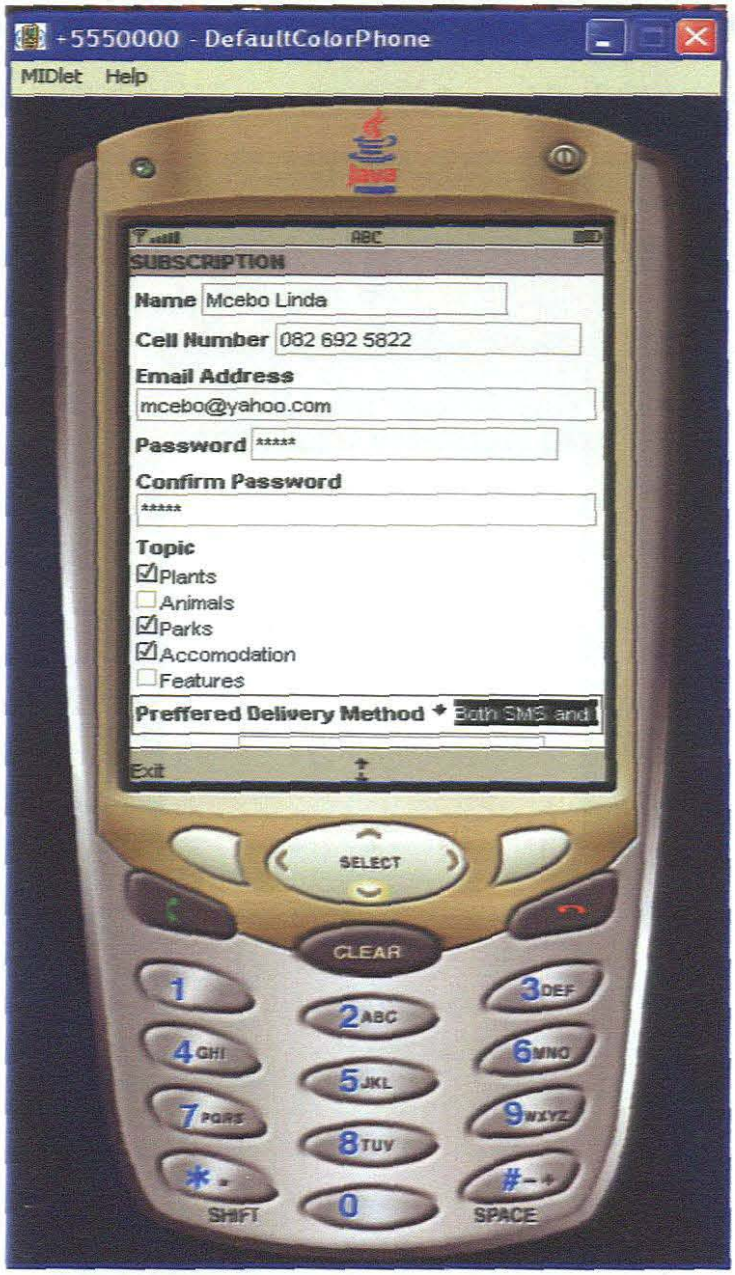

Figure 4. 7: Subscribing on the Mobile Device.

4.2.4 Retrieval of the SANPARKS services information using a mobile device.

The figure 4.8 shows the published SANPARKS service information delivered to the subscriber's mobile device as the notification. The user waits for the notification as the

incoming message to be delivered on his/her mobile device in order to view the SANPARKS services information as a sms on his/her mobile device.

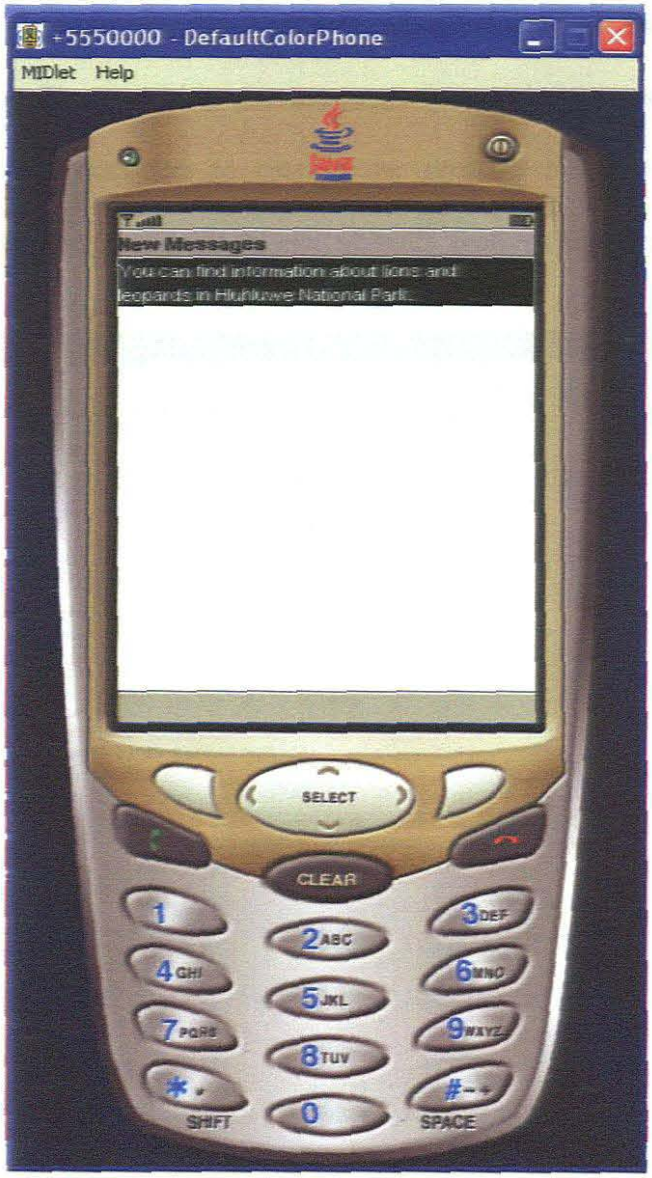

Figure 4. 8: Mobile device showing the services information delivered as a sms.

4.2.5 The Internal Structure of the XML Database.

Figure 4.9 shows the list of subscribers profiles stored on the XML database as XML files. When users subscribe to the SANPARKS Information System their information is

stored as the subscribers' profiles as XML files as shown in the above figure. The information stored in figure 4.9 as the XML structure forms the user profiles of the subscribers and is useful during the matching phase of services information with the relevant subscribers. The matching process is performed by the mediator in order to observe the relevant subscribers to receive the services information that has been published by the information sources (plants, animals, features and accommodation).

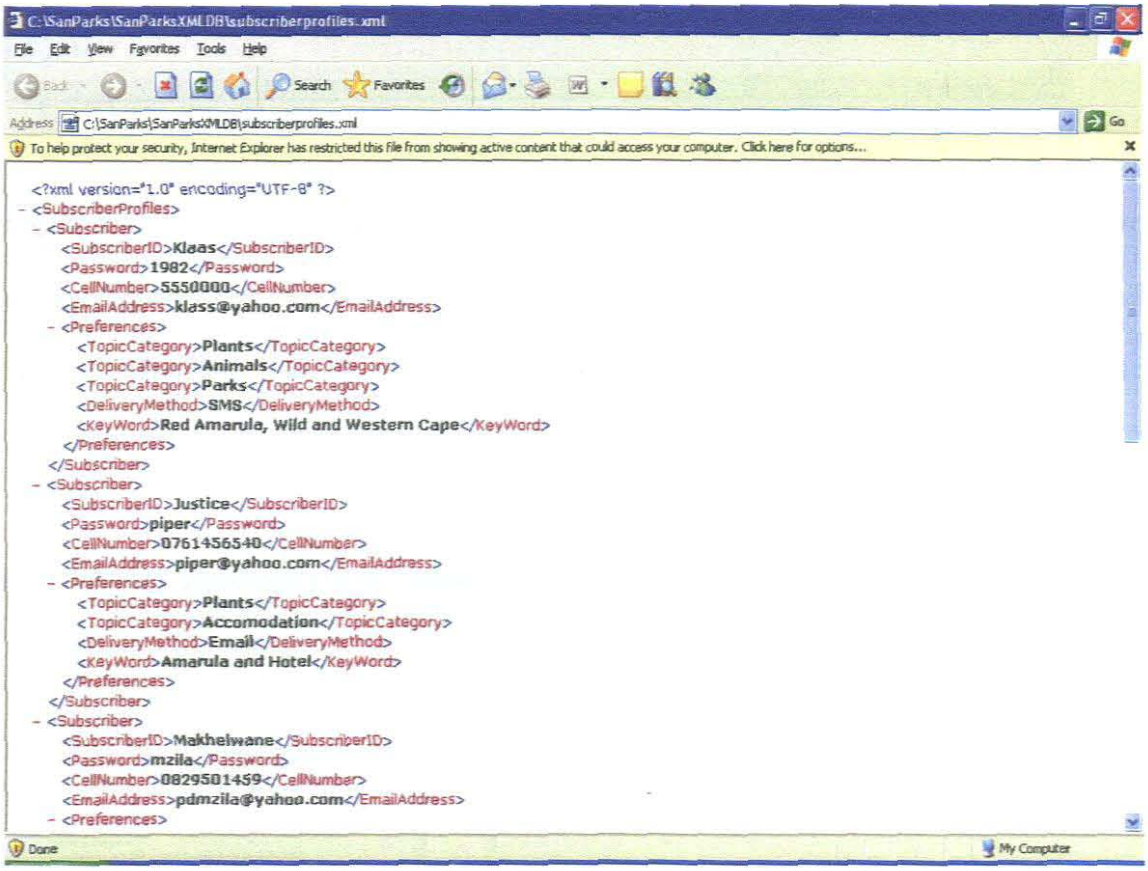

Figure 4. 9: The Subscribers profiles stored on the XML database.

Figure 4.10 shows the XML structure for publisher actors. Publishing is an house-keeping not a public process. A representative of SANPARKS is designated as publisher with respect to each of the four information sources namely plant, animals, features, and accommodation.

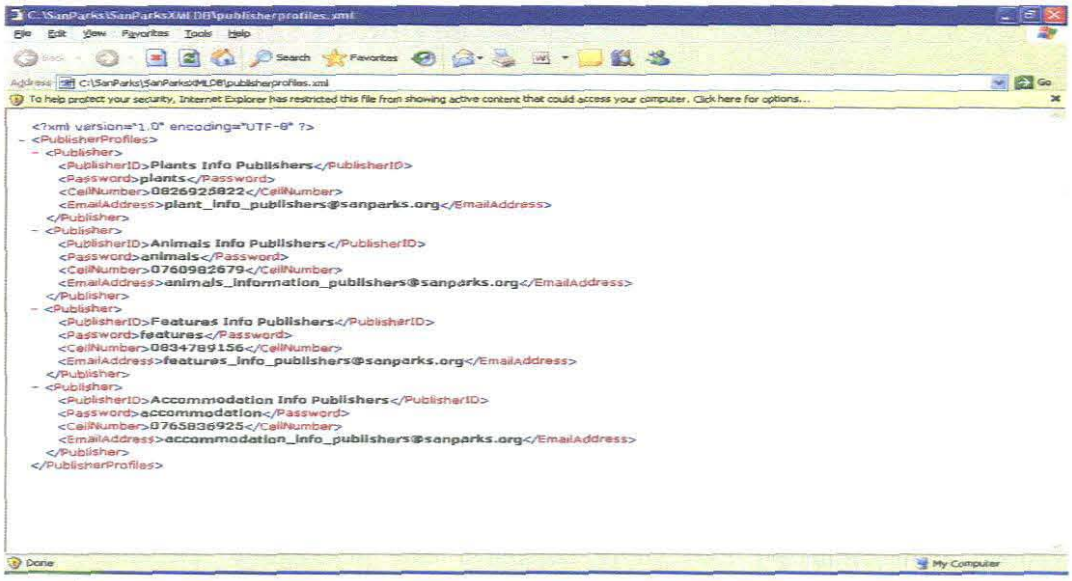

Figure 4. 10: The Publishers profiles on the XML database.

Figure 4.11 shows the list of published messages for the subscribers. These messages are a combination of all the messages sent to either the desktops or mobile devices as e-mail and sms respectively. Every message sent to the subscriber is stored by the system on the XML database as XML files as shown in figure 4.11.

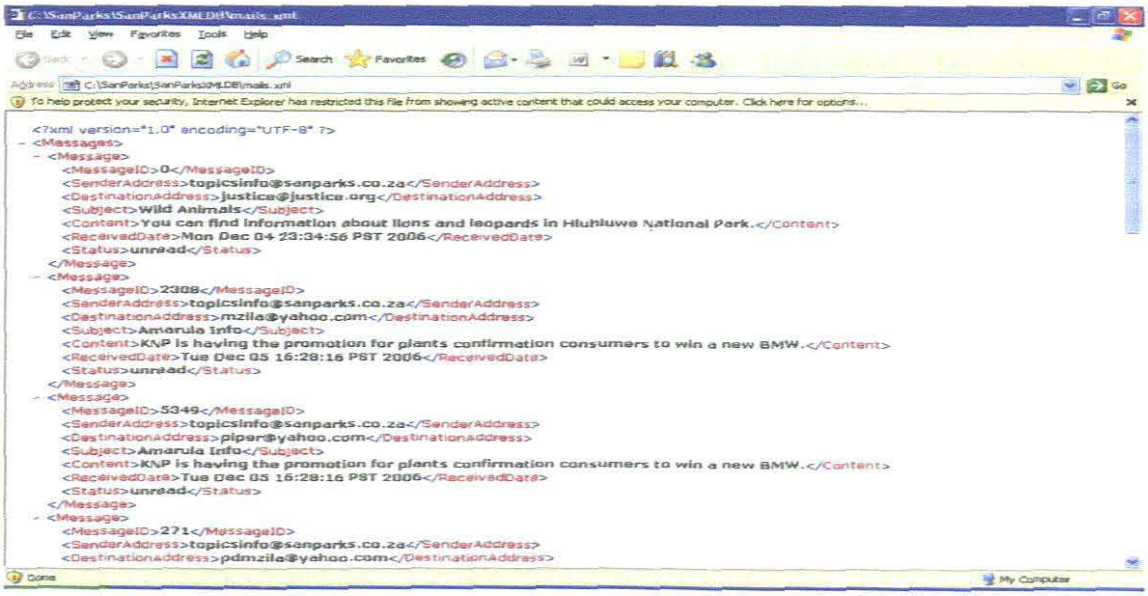

Figure 4. 11: The list of messages sent to the subscribers.

4.3 Usability of the Prototype

4.3.1 Usability Testing.

The model presented in this dissertation is tested for usability. The evaluation of usability is conducted separately for the publisher module and the subscriber module being the two main user-oriented components of the system.

4.3.1 .1 Evaluating the Publisher Component (Technical Evaluation).

The publishers' component was evaluated along four (4) dimensions namely: user friendliness, SANPARKS system fitness to be an information provider entity, SANPARKS system reaching the prospective clients and quality of information provided by the SANPARKS system.

(A) Instrument Design and Administration

The target population for this study consists of students who were from IT-related departments. The Departments are Computer Science, Information Technology and Library Information Systems. Forty students from the above-named departments were sampled randomly from the population. Some interviews were conducted to ascertain the level of commitment of the students.

(B) Analysis and Results

Only thirty of the forty (40) sample students responded amounting to a response rate of 75%. The distribution of the respondents according to Department is shown in Table 4.1:

Table 4. 1: The number of people interviewed as both subscribers and publishers.

Department	Number of People
Computer Science	20
Library & Information Science	2
Information Technology	8
Total	30

Table 4.2 gives the results of publishers' comments when they compared the new SANPARKS system with the old one.

Table 4. 2: The results of publishers comments on the new SANPARKS system.

	Do you notice any changes with the new SANPARKS system compare to the old one?		Total
Department	Major	Few (Minor)	
Computer Science	60.0%	40.0%	100.0%
Library & Information Science	100.0%	0.00	100.0%
Information technology	87.5%	12.5%	100.0%
Total	70.0%	30.0%	100.0%

Table 4.3 shows the results of the publishers' comments on the user friendliness of the new SANPARKS Information System.

Table 4. 3: The publishers comment on the user friendliness of the new SANPARKS System.

	In your opinion is the new system user friendly (or easy to use)?		Total
Department	Yes	Average, normal as other system	
Computer Science	85.0%	15.0%	100.0%
Library & Information Science	100.0%	0.00	100.0%
Information technology	87.5%	12.5%	100.0%
Total	86.7%	13.3%	100.0%

Information provider entity dimension: All respondents say the system is fit for the information provider entity when it delivers notifications to the subscribers via sms and e-mail. Question 5 and 6 of Section A of the questionnaires that were distributed to the information sources was used to determine this dimension. Asking questions 5 and 6 as the same question in different ways helps to eliminate bias. The elimination of bias was done by comparing the results of questions 5 and 6 since these questions have the different results. Comparing these different results helps to determine the results of this dimension.

4.3.1.2 Evaluation of the Subscribers Component (Technical People).

The subscribers' component was evaluated along seven (7) dimension and these are: quality of interaction between the subscribers and the SANPARKS system, quality of information provided by the SANPARKS system to the subscribers, the external features of the SANPARKS system when using it, user friendliness of the SANPARKS system, personalization of services information provided by the SANPARKS system and the limitation of the information provided by the SANPARKS system to the subscribers.

(A) Instrument Design and Administration

The target population for this study consists of students who were from the IT-related departments. The Departments are Computer Science, Information Technology and Library & Information Science. Thirty-five students from the three departments were

sampled randomly from the population. The students were interviewed to get them committed to the evaluation exercise.

(B) Analysis and Results

Thirty of the thirty-five (35) sample students responded making up a response of 86%. The distribution of the respondents according to Department is shown in table 4.1. Table 4.4 gives the results obtained from evaluating the quality of information provided by the SANPARKS system to the subscribers.

Table 4. 4: The results of the quality of information provided to the Subscribers.

		Quality of information provided				
		Good	Fair	Excellent	No-Response	Total
Department	Computer Science	40.0%	10.0%	40.0%	10.0%	100.0%
	Library & Information Science	66.7%	0.00	33.3%	0.00	100.0%
	Information technology	57.1%	0.00	42.9%	0.00	100.0%
Total		46.7%	6.7%	40.0%	6.7%	100.0%

Table 4.5 gives the results of the quality of interaction between the SANPARKS system and subscribers.

Table 4. 5: The results of the quality of interaction between the SANPARKS System and Subscribers.

Department	Quality of interaction			Total
	Good	Fair	Excellent	
Computer Science	55.0%	5.0%	40.0%	100.0%
Library & Information Science	66.7%	0.00	33.3%	100.0%
Information technology	71.4%	0.00	28.6%	100.0%
Total	60.0%	3.3%	36.7%	100.0%

Table 4.6 gives the results of the subscribers after they had used the new system. The subscribers then express their feelings after in use of the new system.

Table 4. 6: The results of using the new SANPARKS System.

	How do you find the SANPARKS system when taking a tour on it (using it)?		Total
Department	Enjoyable	Non-Response	
Computer Science	60.0%	40.0%	100.0%
Library & Information Science	100.0%	0.00	100.0%
Information technology	71.4%	28.6%	100.0%
Total	66.7%	33.3%	100.0%

Table 4.7 shows the subscribers' comments on the user friendliness of the new SANPARKS system.

Table 4. 7: Users opinion on the user friendliness of the Subscriber Module.

Department	In your opinion is the new system user friendly (or easy to use)?			Total
	Yes	No very complicated	Average, normal as other system	
Computer Science	65.0%	0.00	35.0%	100.0%
Library & Information Science	66.7%	0.00	33.3%	100.0%
Information technology	71.4%	14.3%	14.3%	100.0%
Total	66.7%	3.3%	30.0%	100.0%

Table 4.8 shows the results of the subscribers' comments about the service information offered by the new SANPARKS system.

Table 4. 8: The results of the subscribers' comments about the services provided by the new SANPARKS System.

	Does the SANPARKS system satisfy your interest on the services?		Total
Department	Yes	No	
Computer Science	95.0%	5.0%	100.0%
Library & Information Science	100.0%	0.00	100.0%
Information technology	100.0%	0.00	100.0%
Total	96.7%	3.3%	100.0%

The figure 4.12 is the results of the subscribers' views on the information provided by the new SANPARKS system. The respondents had two options when answering the questionnaire. They were to select either 'Yes' or 'No', if 'Yes' this specified what needed to be improved. When the subscriber selected 'No' this meant that the information provided by the new SANPARKS information system did not provide limited information. When selecting yes this meant that new system provided limited information which needs to be improved. The subscriber was provided with ample space in the questionnaire to specify the improvements.

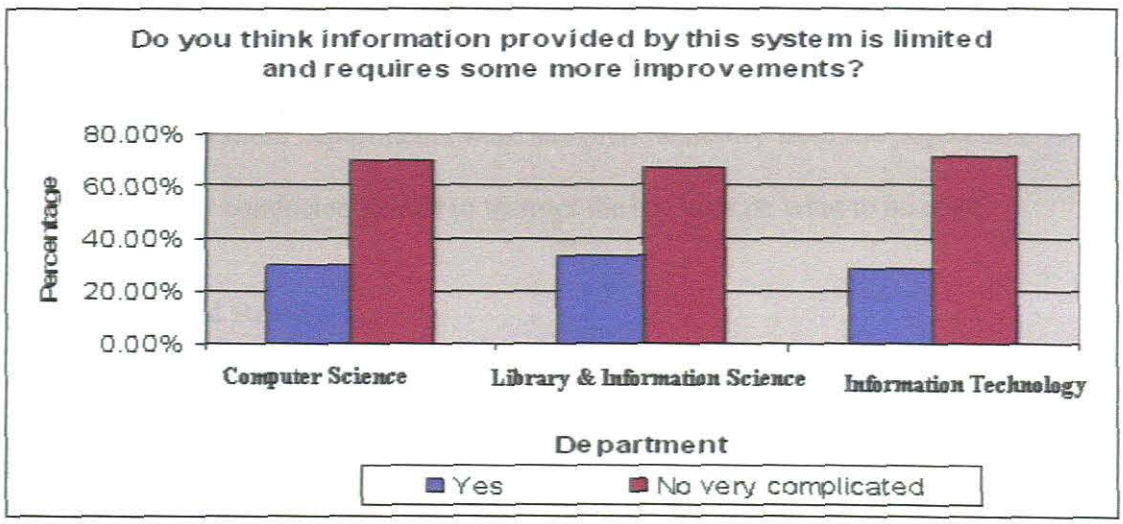

Figure 4. 12: Quality of the information provided to subscribers.

Personalization Dimension: The aspect of personalization that is measured in this research is the personalization of services information delivered to the subscribers by push style. The questionnaire that was distributed to subscribers was targeting to find out whether the information disseminated to the subscribers is simplified and personalized. The question number 7 of section A of the questionnaires was used to observe the views of the information consumers (subscribers). The result of personalization dimension is

based on whether the subscribers receive the services information that they subscribe for. All respondents said that the system supported personalization of services when it delivers notifications to them in terms of sms and e-mail.

4.3.1.3 Evaluation by barely IT-literate.

This evaluation was conducted along two (2) dimension and these are: user friendliness and the behavior of the SANPARKS system when using it.

(A) Instrument Design and Administration

The target population for this study consisted of students who have little or no IT background. They were sampled from the departments of Psychology and Social Work. 35 students from these departments were sampled randomly from the population. A one-on-one interview conducted helped to instruct the students on what to be done.

(B) Analysis and Results

Twenty nine of the thirty five sampled students responded by making up a response of 83%. The results of the non-technical that were interviewed is shown in figure 4.13 together with their feelings of the new SANPARKS Information system.

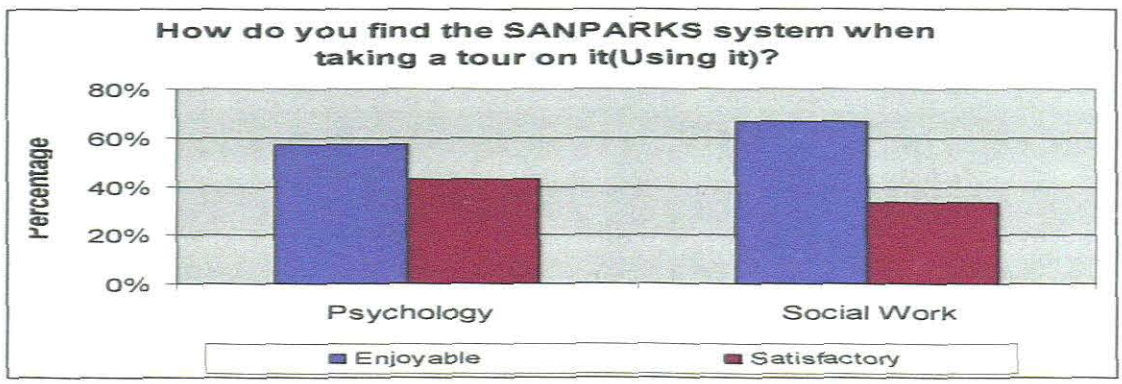

Figure 4. 13: The results of Arts students interviewed for user-friendliness of the SANPARKS System.

The Figure 4.14 gives the results of the non-technical people opinions after they have used the new SANPARKS Information System. About 65% of the students from psychology and about 80% of those from Social work found the system to be user friendly.

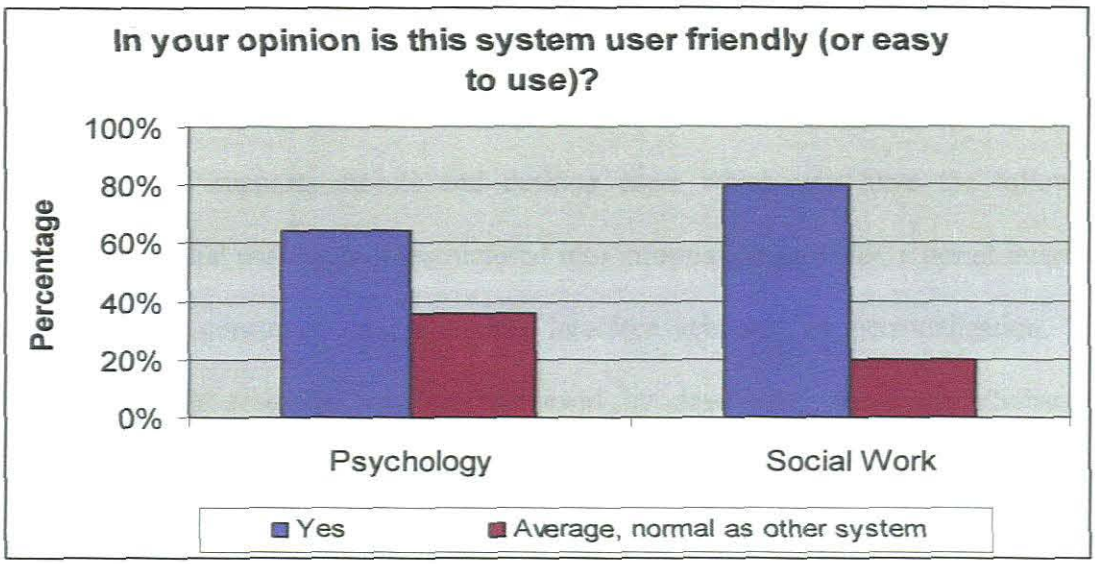

Figure 4. 14: The results of Arts students interviewed for user-friendliness of the SANPARKS System.

A comparison of the results depicted in Table 4.7 and Figure 4.14 reflects that the system was found to be considerably user friendly by both students from the IT related and non IT related departments.

CHAPTER 5

CONCLUSION

5. 1 Conclusion

The objective of this research was primarily to develop a Publish/Subscribe Architectural Framework that supports mobile and desktop users which will have the following features: a national park system restructured into information provider, a portal interface for information consumers and a mobile interface achieved by personalization. The objective of this research has been achieved by developing the Publish/Subscribe Architectural Framework that supports mobile and desktop users among other features. The features that were achieved in order to fulfill the objective of this research are the national park system restructured into information provider, a portal interface for information consumers and a mobile interface achieved by personalization.

A restructured SANPARKS system has been achieved by the support of the new conceptualized model and the system architecture that ensures that subscribers receive information without knowing the source. A portal interface that has been achieved allows the information consumers to subscribe to the SANPARKS system as subscribers. Subscribers then have an access to view the personalized SANPARKS services information as emails using the new developed SANPARKS portal interface. The mobile interface has also been developed which allows the subscribers to view personalized SANPARKS services information as sms or mms using their mobile devices.

Moreover, an improved SANPARKS Information System that personalizes user's subscription is implemented. The system as implemented does not consider the issue of context awareness, which is very crucial in a mobile computing environment during the process of pushing messages to the user. The Context Awareness requirement can be met by incorporating the research results of another work being done in the Department of Computer Science, University of Zululand. (Jembere, et al., 2006).

Persistence of information is very crucial in this study in order to ensure mobile users do not loose information when temporarily disconnected from the network. The scope of this work is limited to push information to people who are connected to the network with their mobile devices, if the mobile device reconnects to the network it does not receive notification of the messages that were sent during disconnection time. The implementation serves only as a proof of concept of the publish/subscribe idea works even without supporting persistence during intermittent network disconnections. Finally the objectives that have been achieved on this work affirm that the goal of the research mentioned in chapter one has been reasonably achieved.

5.2 Contributions

The dissertation attempts to show that SANPARKS can be restructured to operate as a distributed information service provider. Using the publish/subscribe communication paradigm, consumers who have interest in information that SANPARKS specialize in can be reached without using interactive approach. Once identified only interested consumers receive information, from SANPARKS, as they become available. In this way, the system is designed to support personalization.

5.3 Future Work

To implement a successful persistent information delivery service/engine there is a need to adopt JMS (Java Message Services) that uses topics to temporally store the published services information during disconnection time. This would assure the delivery of information sent during disconnection time to the subscribers when they reconnect to the network. This would also motivate the subscribers to keep accessing the SANPARKS services information as they have an easy way of getting information even those sent when they are not connected to the network.

This study is based on software but it can also be extended to the networking environment by adopting the same publish/subscribe communication paradigm for content dissemination. This would enable the researcher to identify some algorithm that would be tested with the following metrics: minimal processing load, minimal bandwidth consumption and notification delay to solve the performance of the publish/subscribe systems in a dynamic environment especially with mobile clients.

This research could also include personal mobility as the design principle that would allow service users to publish and receive the content using various terminals in different networks. This feature would enable true personal mobility and offer usage flexibility.

APPENDIX A

USER MANUAL

This part of the thesis describes the necessary step to using the SANPARKS Information System in both the desktop and the mobile device.

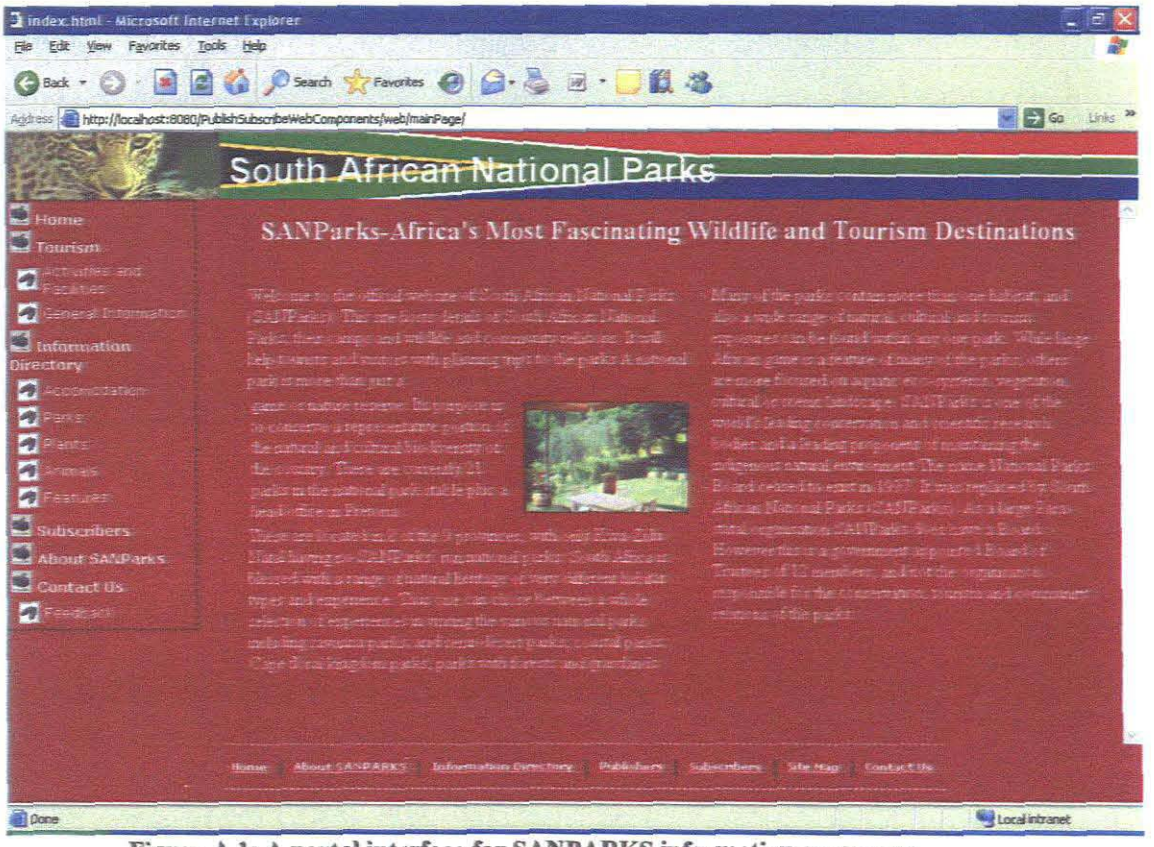

Figure A.1: A portal interface for SANPARKS information consumers.

The figure A.1 is used by the information consumers to take a tour on the SANPARKS Information system in order to familiarize themselves with the services information provided by this system. The information consumers can be able to view the information provided by the system by clicking the links that shows up on the left of the portal except the subscribers link because for the consumer to access this link they need to be registered or subscribed to the SANPARKS Information system using the same portal.

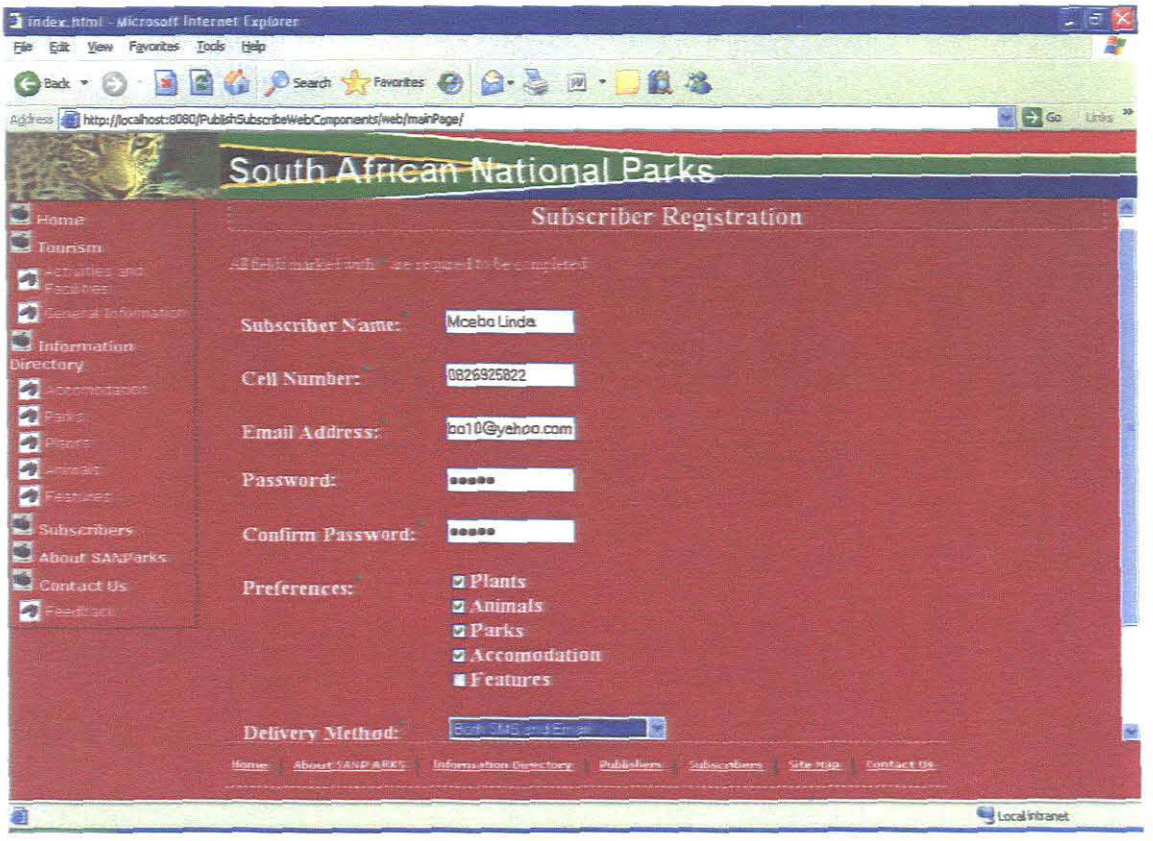

Figure A.2: Subscriber's Registration Interface on the SANPARKS Portal.

The figure A.2 is used by the clients to subscribe or register with the SANPARKS system, in order to be authenticated by the system next time when they want to access the subscribers' services information offered by this system.

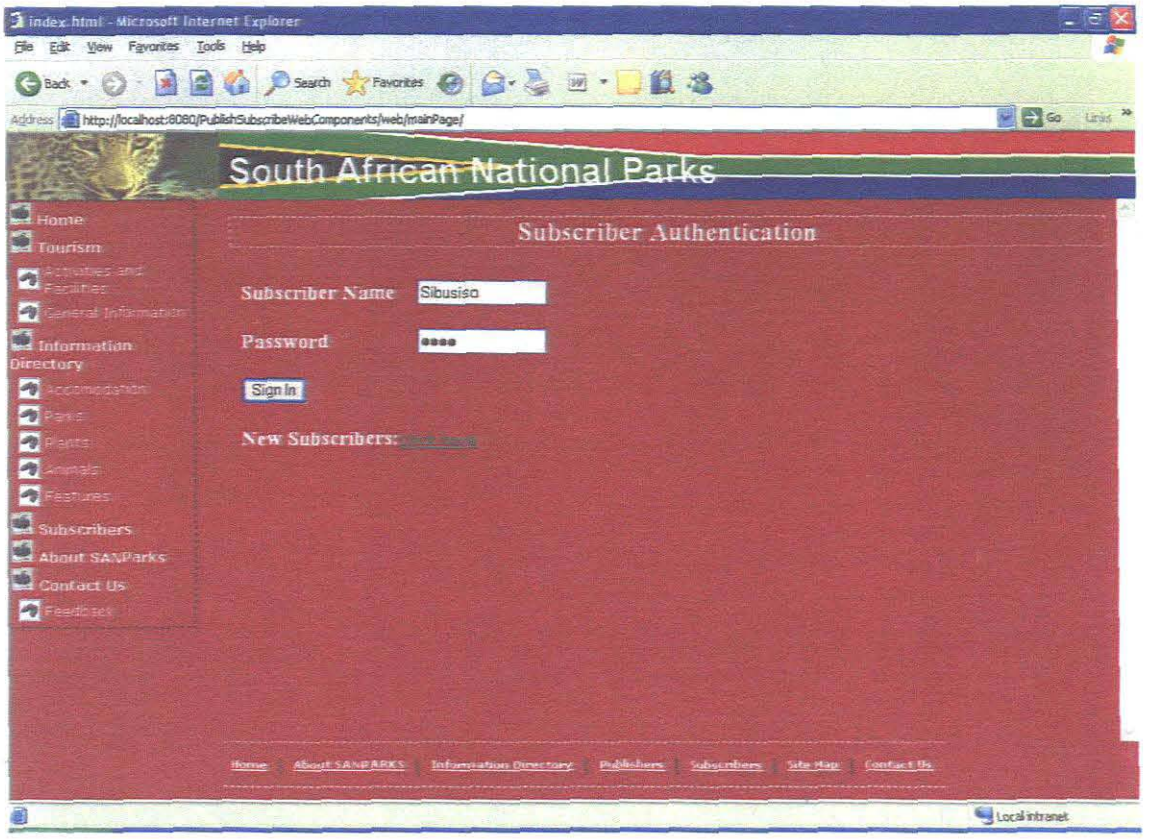

Figure A.3: Subscriber's Authentication Interface on the SANPARKS portal.

This figure A.3 is used by the subscribers for the system authentication. The subscribers provide the information they have used when they were subscribing to the system. The information they provide is the user name and the passwords.

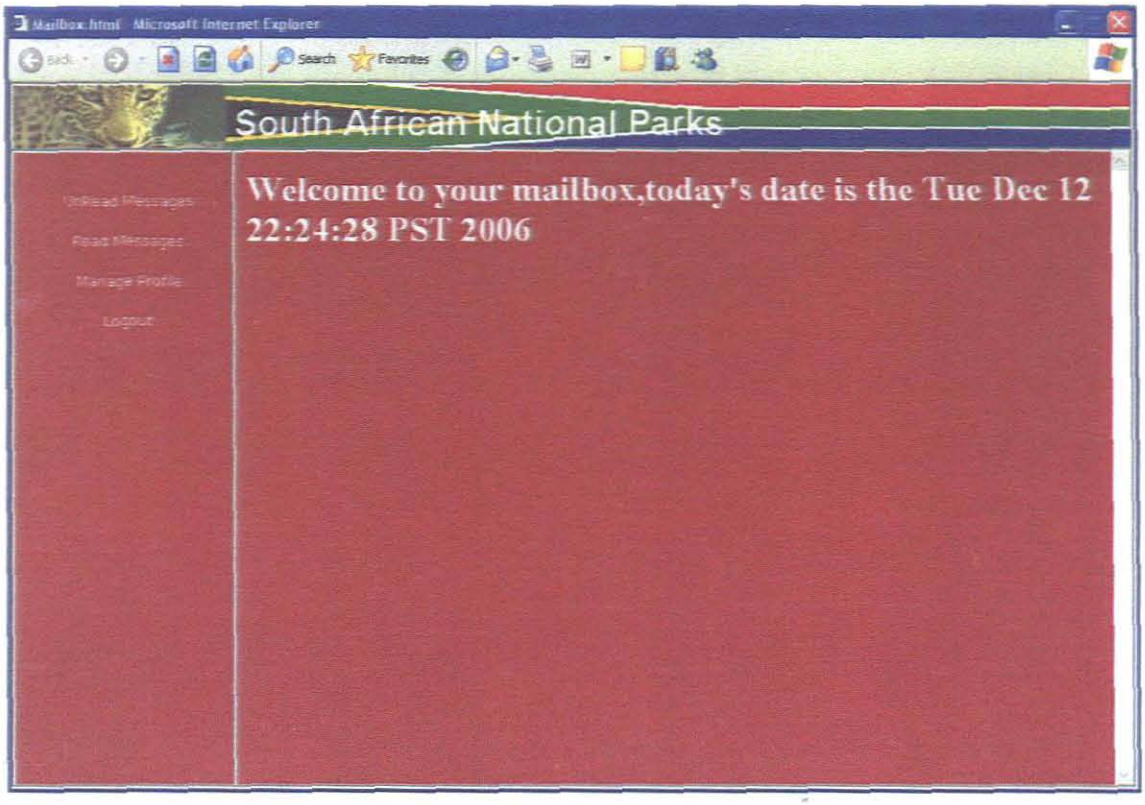

Figure A.4: Desktop Welcoming Interface for Subscribers to access SANPARKS services information published as E-mails.

This figure A.4 is used by the subscribers when they want to access the published services information. The subscribers has a choice of reading the unread message as new send e-mail by clicking on the unread message link provided on the left of the figure A.4 by the system.

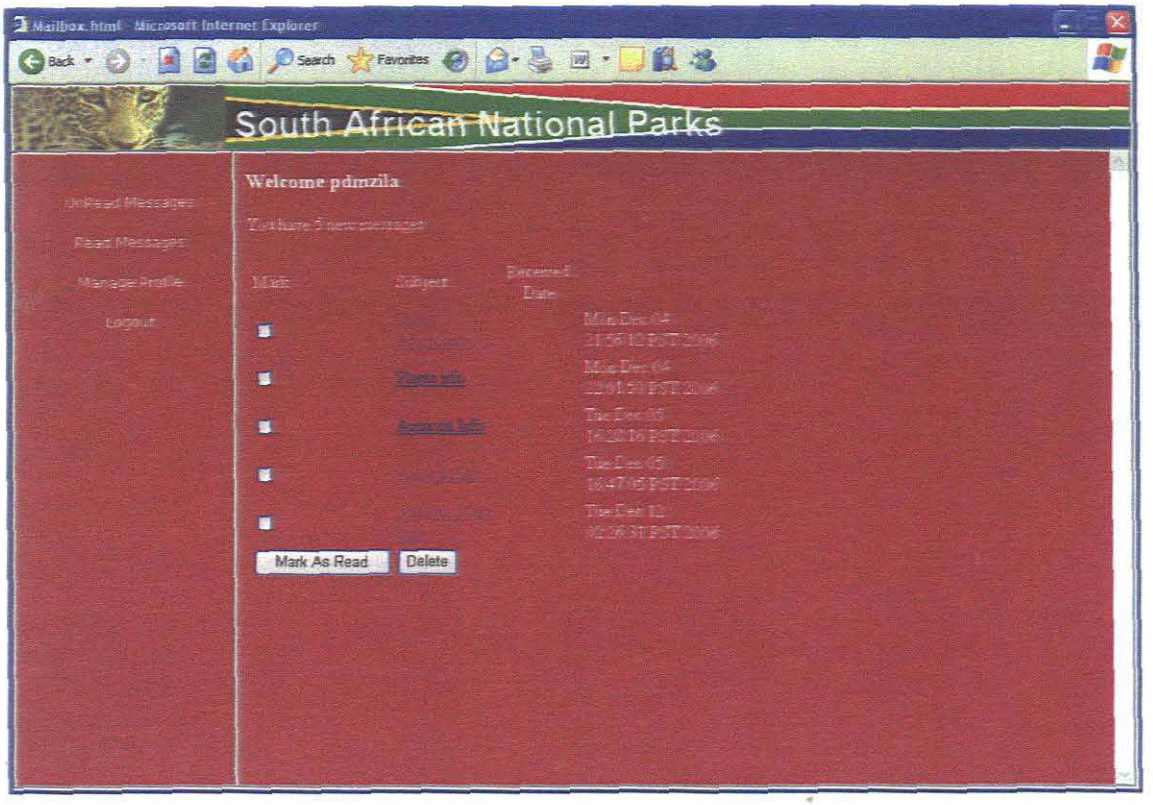

Figure A.5: The list of E-mails on the Desktop.

The figure A.5 shows the list of e-mails send as the new messages. The subscribers can click the links appearing on the figure under subject in order to gain access to the information that has been published. When the subscribers want to delete the messages that they have been reading, they can do so by marking the message(s) they want to delete and then after click the delete button appearing on figure A.5. They can also move the same messages to the read folder, by marking the message(s) they want to move to the read folder and then after click the Mark As Read button appearing on figure A.5. They can even read the message moved to the read folder by clicking the Read Messages link also appearing on the left of figure A.5. Subscribers can also manage their profiles by deleting or updating them when they click the Manage Profile link on the same figure A.5 to finish the process of managing their profiles. Each and every time subscribers

perform one of the processes like moving a message or to Read Message folder, delete messages they have been reading, managing profiles and log outing complete from the system they are notified by the system whether they are successful or not in the performance of their operation.

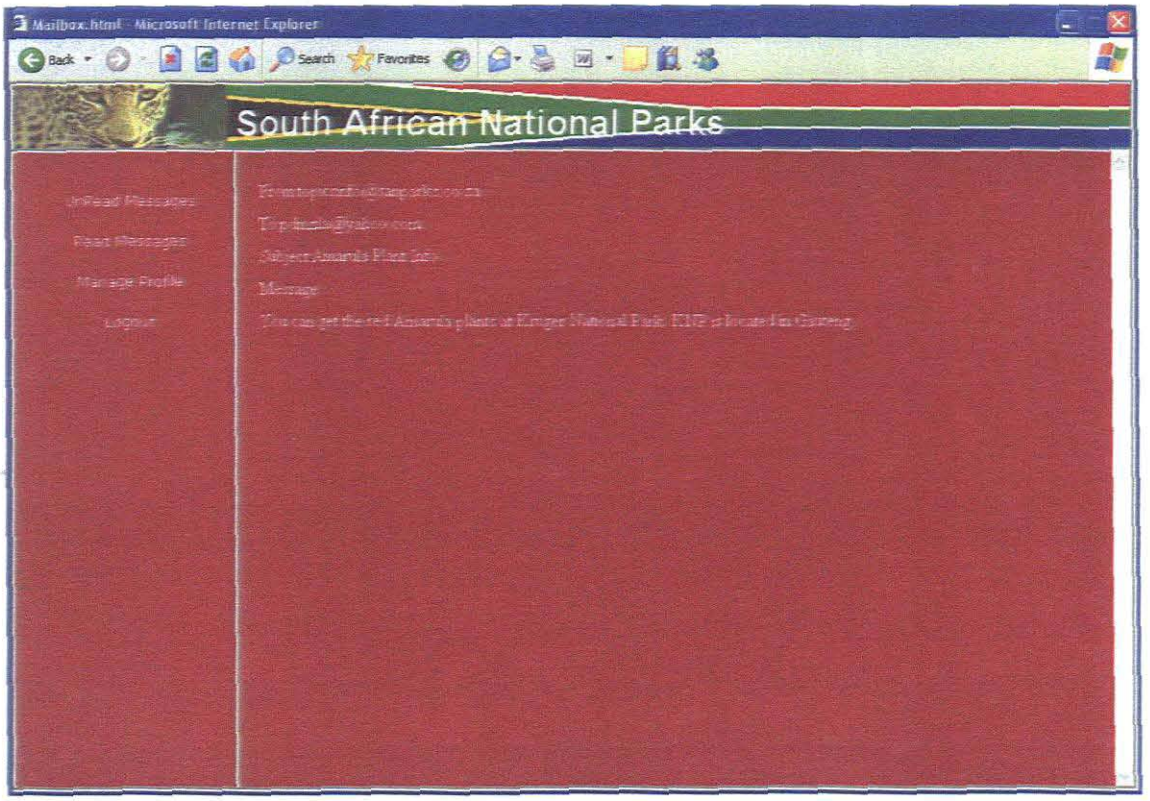

Figure A.6: The service information delivered as the e-mail.

The figure A.6 shows the service information delivered on the desktop as e-mail. Subscribers use this figure to read the messages send as e-mail in order to gain access to the services information.

Figure A.7: Welcoming Interface on the Mobile Device.

The figure A.7 shows the welcoming screen of the client's mobile device. The above figure A.7 shows that the clients can have two option of gaining access to the SANPARKS services information. The new clients need to subscribe with the SANPARKS Information by selecting subscribe option on their mobile device, then after provides the information required by the mobile device in figure A.8. The subscribers as people who already subscribe with the SANPARKS system need to make new messages option to show that they are interested to gain access to the new messages being sent as notification.

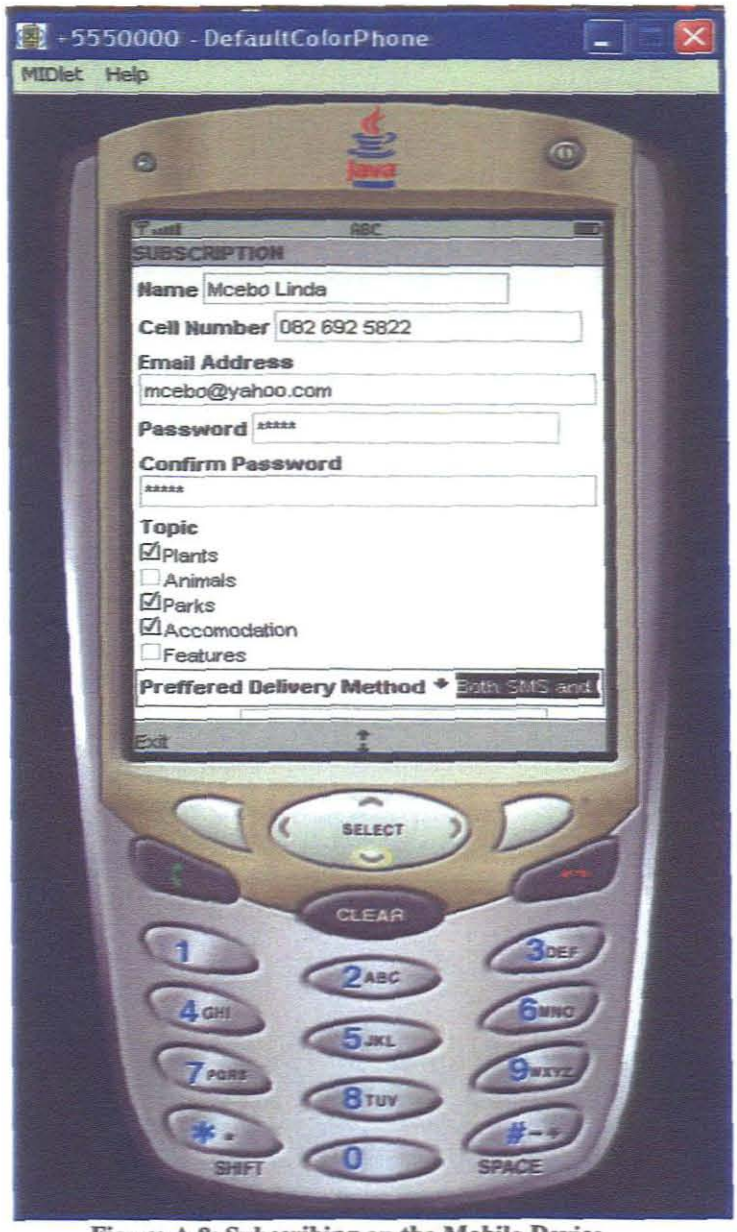

Figure A.8: Subscribing on the Mobile Device.

The figure A.8 shows that the information consumers can use their mobile devices to subscribe for the SANPARKS services information to be delivered as sms on their mobile devices or e-mail on the desktop. The subscribers need to provide all the required information by filling the text fields provided by the mobile device in figure A.8 in order to be successful in registering with the SANPARKS Information System.

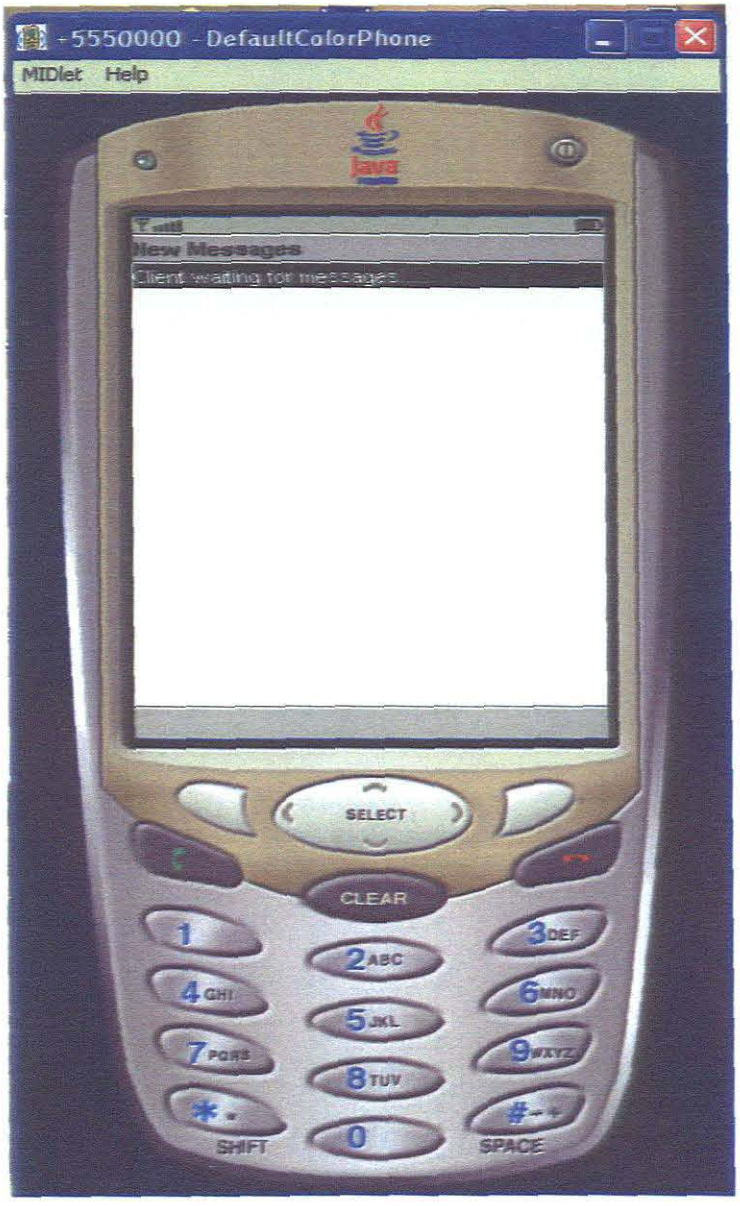

Figure A.9: Mobile device on waiting state for the incoming messages.

The figure A.9 shows the clients mobile device on the waiting state of the incoming message(s). The delivered message(s) on the mobile device can be accessed by the mobile clients as shown in figure A.10.

Figure A.10: Mobile device showing published information.

The figure A.10 shows the services information delivered on the mobile device as the sms. This service information serves as the notification to the subscribers in order to be aware of the services information available on the SANPARKS organization.

APPENDIX B

UML DESIGN DOCUMENTATION

CLASS DIAGRAM AND DESCRIPTION

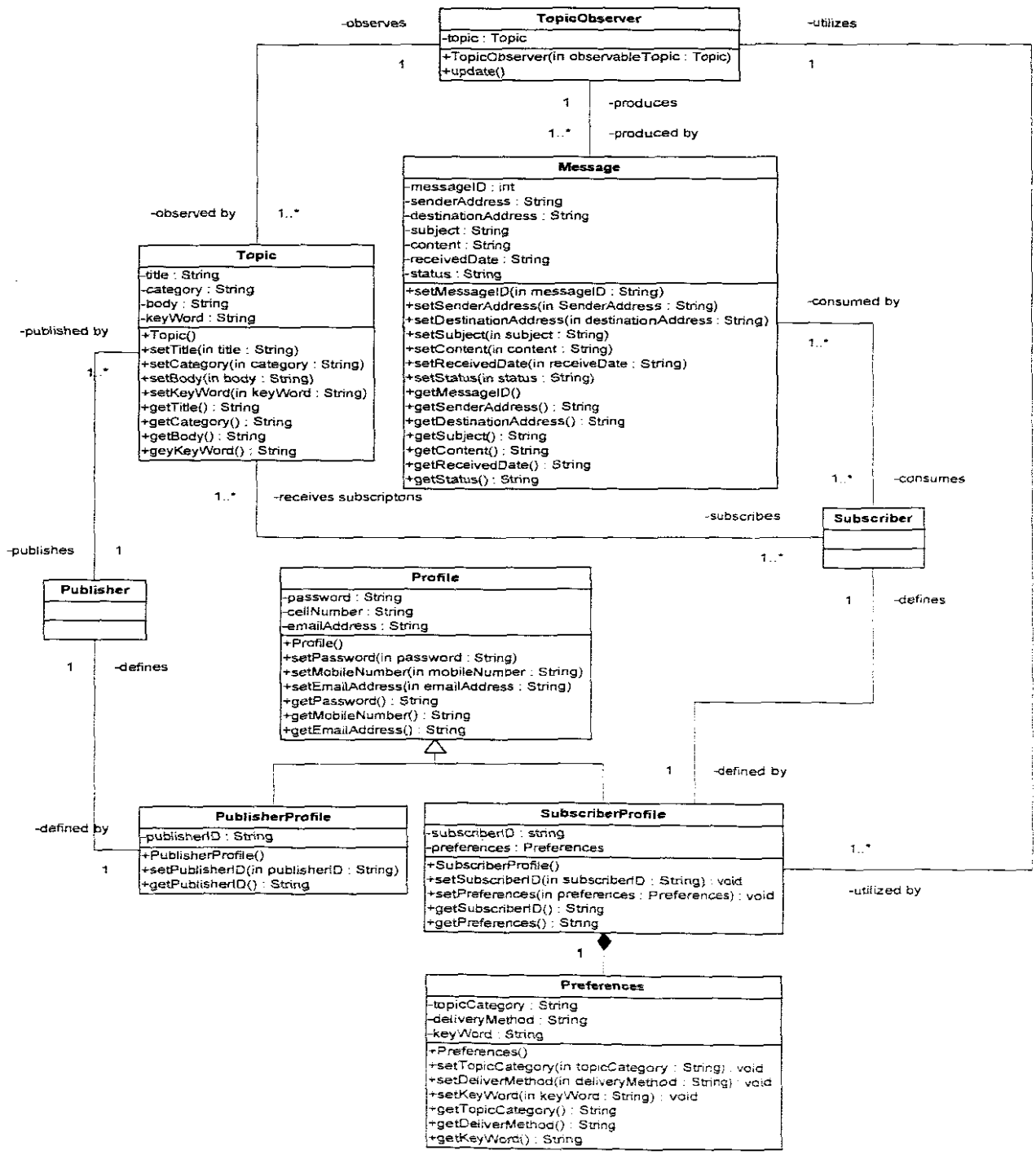

Figure B.1 Class Diagram of the SANPARKS Information System Implementation.

Classes Definition

Publisher: This class is represented by a bean that defines properties and behaviors for handling information of the publisher entity.

Subscriber: This class is represented by a bean that defines properties and behaviors for handling information of the subscriber entity.

Message: This class is responsible for storing messages in an XML storage format which acts as a mailbox for emails. The XML storage is only for subscribers who chose email as a delivery method. The Message class defines methods for saving, deleting and retrieving emails. Furthermore the class defines methods for marking read messages.

TopicObserver: This class is responsible for listening to messages published to a topic and then sends the message to potential subscribers when publication is received. This class applies the Observer design pattern in modeling collaboration between system objects.

Topic: This class represents a bean that defines properties and behaviors for handling topic publications. This class is a convergence of four properties the title, category, body and a keyword.

Profile: This class represents a generic class for both the subscriber's profile and the publisher's profile. Furthermore it is a bean class with defined properties and behaviors that are supposed to be shared among its descendant.

SubscriberProfile: This class is a derivation from the super class Profile. Furthermore it is a specialized bean that defines properties and behaviors for handling information pertinent to the subscriber profile.

PublisherProfile: This class is a derivation of the super class Profile.Basically it is a specialized bean that defines properties and behaviors for handling information pertinent to the publisher profile.

Preferences: This is a delegate in class SubscriberProfile and it is responsible for capturing and querying subscriber preferences. It defines attributes that enable subscriber preferences to be persisted to permanent storage.

APPENDIX C

SOURCE CODE

Mobile Client

Subscribing on the mobile device

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;


public class Subscribe extends MIDlet implements CommandListener, ItemCommandListener
{
        //Commands
        private static final Command CMD_SUBMIT = new Command("Press",Command.ITEM,1);
        private static final Command CMD_RESET  = new Command("Press",Command.ITEM,1);
        private static final Command CMD_EXIT = new Command("Exit",Command.EXIT,1);
        private static final Command CMD_OK = new Command("OK",Command.OK,1);

        private List confirmScreen;

        //Subscription form item commands
        private StringItem submit;
        private StringItem reset;

    //Subscription form
        private Form frmSub;


        //Subscription form textfields
        private TextField txtSubscriberID;
        private TextField txtCellNumber;
        private TextField txtEmailAddress;
        private TextField txtPassword;
        private TextField txtConfirmPassword;

        //Subcription form checkboxes
        private ChoiceGroup topicList;
        private ChoiceGroup cgDeliveryMethod;
        private TextField txtKeyWord;

        //Application Screen
        private Display display;

        //Welcome String
         private Alert errorAlert;
         FieldValidator verifier;

        public Subscribe()
        {

     verifier = new FieldValidator();
     //SUBSCRIPTION
     txtSubscriberID = new TextField("Name","",15,TextField.ANY);
     txtCellNumber = new TextField("Cell Number","",15,TextField.PHONENUMBER);
     txtEmailAddress = new TextField("Email Address","",15,TextField.EMAILADDR);
     txtPassword = new TextField("Password","",15,TextField.PASSWORD);
     txtConfirmPassword = new TextField("Confirm Password","",15,TextField.PASSWORD);
          txtKeyWord = new TextField("KeyWord","",15,TextField.ANY);

     String [] topics = {"Plants","Animals","Parks","Accomodation","Features"};
```

```
    Image [] topicImages = null;
    topicList = new ChoiceGroup("Topic",Choice.MULTIPLE,topics,topicImages);

    String [] methodChoices = {"Select a delivery method","SMS","Email","Both SMS and Email"};
    Image [] methodImages = null;
    cgDeliveryMethod =  new ChoiceGroup("Preffered Delivery Method",ChoiceGroup.POPUP,methodChoices,methodImages);



    //Item commands initialization and event registration

    submit = new StringItem("","Submit",Item.BUTTON);
    submit.setDefaultCommand(CMD_SUBMIT);
    submit.setItemCommandListener(this);
    reset = new StringItem("","Reset",Item.BUTTON);
    reset.setDefaultCommand(CMD_RESET);
    reset.setItemCommandListener(this);

   //form initialization and event registration

    frmSub = new Form("SUBSCRIPTION");
    frmSub.append(txtSubscriberID);
    frmSub.append(txtCellNumber);
    frmSub.append(txtEmailAddress);
    frmSub.append(txtPassword);
    frmSub.append(txtConfirmPassword);
    frmSub.append(topicList);
    frmSub.append(cgDeliveryMethod);
    frmSub.append(txtKeyWord);
    frmSub.append(submit);
    frmSub.append(reset);
    frmSub.addCommand(CMD_EXIT);
    frmSub.setCommandListener(this);

     errorAlert = new Alert("Error");
          errorAlert.setType(AlertType.ERROR);
          errorAlert.setTimeout(1000);

     confirmScreen =  new List("Subscription Confirmation.....",Choice.IMPLICIT);
          confirmScreen.addCommand(CMD_OK);
          confirmScreen.setCommandListener(this);


}
public void startApp()
{

          getDisplay().setCurrent(frmSub);

}

public void pauseApp()
{
}

public void destroyApp(boolean unconditional)
{
}
public void commandAction(Command c,Item item)
{
     if(c == CMD_SUBMIT)
     {
          Thread t = new Thread()
          {
               public void run()
               {
                    try
                    {
                         String subscriberID  = txtSubscriberID.getString().trim();
```

```
String cellNumber = txtCellNumber.getString().trim();
String emailAddress = txtEmailAddress.getString().trim();
String password = txtPassword.getString().trim();
String conpassword = txtConfirmPassword.getString().trim();
String [] topics = new String[topicList.size()];

for(int i = 0;i < topics.length;i++)
{

    if(topicList.isSelected(i)
    {
        topics[i] = topicList.getString(i);
    }
}
String deliveryMethod =
(cgDeliveryMethod.getString(cgDeliveryMethod.getSelectedIndex())).trim();
String keyWord = txtKeyWord.getString();

if(verifier.checkEmptyID(subscriberID))
{
        errorAlert.setString("ID Required");
        getDisplay().setCurrent(errorAlert,frmSub);
}
else if(verifier.checkEmptyCellNumber(cellNumber))
{
        errorAlert.setString("CellNumber Required");
        getDisplay().setCurrent(errorAlert,frmSub);
}
else if(verifier.checkEmptyEmailAddress(emailAddress))
{
        errorAlert.setString("Email Required");
        getDisplay().setCurrent(errorAlert,frmSub);
}
        else if(verifier.checkEmptyPassword(password))
{
        errorAlert.setString("Password Required");
        getDisplay().setCurrent(errorAlert,frmSub);
}
else if(verifier.checkEmptyConPassword(conpassword))
{
    errorAlert.setString("Please confirm password");
    getDisplay().setCurrent(errorAlert,frmSub);
}
        else if(verifier.checkNonSelectedTopics(topics))
{
        errorAlert.setString("Please select a topic");
        getDisplay().setCurrent(errorAlert,frmSub);
}
        else
if(verifier.checkNonSelectedDeliveryMethod(cgDeliveryMethod.getSelectedIndex()))
{
        errorAlert.setString("Please select a delivery method");
        getDisplay().setCurrent(errorAlert,frmSub);
}
        else if(verifier.confirmPassword(password,conpassword))
{
        if(verifier.checkEmptyKeyWord(keyWord))
        keyWord = null;
        if(verifier.validateEmailAddress(emailAddress)&&
verifier.validatePhoneNumber(cellNumber))
{
        String validID = verifier.processEmptyChars(subscriberID);
        String validCell = verifier.processEmptyChars(cellNumber);
        String validEmail = verifier.processEmptyChars(emailAddress);
        String validPassword = verifier.processEmptyChars(password);
        String validMethod = verifier.processEmptyChars(deliveryMethod);
        String validKeyWord = verifier.processEmptyChars(keyWord);
String response =
subscribe(validID,validPassword,validCell,validEmail,topics,validMethod,validKeyWord);
if(response.trim().equalsIgnoreCase("success"))
```

```
                    {
                        Image img = null;
                        confirmScreen.append("Registration completed",img);
                        getDisplay().setCurrent(confirmScreen);
                    }
                        else
                    {
                        errorAlert.setString("Registration failed");
                        getDisplay().setCurrent(errorAlert,frmSub);
                    }
                      }
                        else if(!verifier.validateEmailAddress(emailAddress))
                     {
                        errorAlert.setString("Invalid Email");
                        getDisplay().setCurrent(errorAlert,frmSub);
                    }
                        else if(!verifier.validatePhoneNumber(cellNumber))
                    {
                        errorAlert.setString("Invalid CellNumber");
                        getDisplay().setCurrent(errorAlert,frmSub);
                    }
                        }
                        else
                        {
                        errorAlert.setString("Password mismatch");
                        getDisplay().setCurrent(errorAlert,frmSub);
                    }
                     }
                        catch(Exception ex)
                     {
                        ex.printStackTrace();
                    }
                      }
                        };
                        t.start();
                    }
                        if(c == CMD_RESET)
                {
                        txtSubscriberID.setString(null);
                        txtPassword.setString(null);
                        txtConfirmPassword.setString(null);
                        txtCellNumber.setString(null);
                        txtEmailAddress.setString(null);

        }

  }


public void commandAction(Command c,Displayable d)
{
        if(c == CMD_EXIT)
        {
                destroyApp(false);
                notifyDestroyed();
        }
        if(c== CMD_OK)
        {
                getDisplay().setCurrent(frmSub);
        }
  }
  public Display getDisplay()
  {
        return Display.getDisplay(this);
  }
public String subscribe(String subscriberID,String password,String cellNumber,String emailAddress,String [] topics,String
deliveryMethod,String keyWord)
{
```

```
System.out.println(subscriberID);
System.out.println(password);
System.out.println(cellNumber);
System.out.println(emailAddress);
for(int i = 0;i < topics.length;i++)
            System.out.println(topics[i]);
System.out.println(deliveryMethod);
System.out.println(keyWord);

 StringBuffer buf = new StringBuffer();

 String p1 = "http://localhost:8080/PublishSubscribeWebComponents/SubscriptionHandler" + "?" +
"subscriberID=" + subscriberID + "&" +
"password=" + password + "&" +
"cellNumber="+ cellNumber + "&" +
"emailAddress=" +emailAddress + "&";

buf.append(p1);
for(int i = 0;i < topics.length;i++)
{
        if(topics[i] != null)
                buf.append("topic=" + topics[i]+ "&" );
}
 String p2 = "deliveryMethod=" + deliveryMethod+ "&"+
"keyWord="+ keyWord;
buf.append(p2);

 String uri = buf.toString();

//System.out.println(URI);

HttpConnection connector = null;
DataOutputStream out = null;
String response = null;
try
{
connector = (HttpConnection)Connector.open(uri);
connector.setRequestMethod(HttpConnection.GET);
connector.setRequestProperty("User-Agent","Configuration/CLDC 1.1 Profiles/MIDP 2.0");


        //Writing to a servlet

        int rc = (int)connector.getResponseCode();
         if(rc  !=  HttpConnection.HTTP_OK)
         {
                throw new IOException("Response code not ok");
         }

        response = getServerResponse(connector);
}
catch(Exception ex)
{
        ex.printStackTrace();
}
finally
{
        try
        {
                if(connector != null)
                {
                        out.close();
                        connector.close();
                }
        }
        catch(Exception ex)
        {
                ex.printStackTrace();
        }
```

```
                }
                return response;

        }

        //Getting the servlet response
        public String getServerResponse(HttpConnection connector)
        {
                DataInputStream in = null;
                String response = null;
                try
                {
                        in  = connector.openDataInputStream();
                        int contentLength = (int)connector.getLength();
                        byte [] data  = new byte[contentLength];
                        int length = in.read(data);
                        response = new String(data,0,length);

                }
                catch(Exception ex)
                {
                        ex.printStackTrace();
                }
                finally
                {
                        try
                        {
                                if(connector != null)
                                {
                                        in.close();
                                        connector.close();
                                }
                        }
                        catch(Exception ex)
                        {
                                ex.printStackTrace();
                        }
                }
                return response;
        }


}
```

Receiving messages on a mobile device

```
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.wireless.messaging.*;

import java.io.IOException;


public class SMSReceive extends MIDlet implements CommandListener, Runnable, MessageListener
{

        Command exitCommand  = new Command("Exit", Command.EXIT, 2);
        Command okCommand = new Command("Read Text Messages", Command.OK, 1);
        Alert content;
        Display display;
        Thread thread;
        String[] connections;
        boolean done;
        String smsPort;
```

```
MessageConnection smsconn;
Message msg;
String senderAddress;
Image im = null;
 Displayable s;
List output;
public SMSReceive()
{

        output = new List("New Messages",Choice.IMPLICIT);
        smsPort = "5000";
        s = output ;
        display = Display.getDisplay(this);


}
public void startApp()
{

        String smsConnection = "sms://:" + smsPort;
        if (smsconn == null)
        {
                try
                {
                        smsconn = (MessageConnection) Connector.open(smsConnection);
                        smsconn.setMessageListener(this);
                }
                catch (IOException ioex)
                {
                        ioex.printStackTrace();
                }
        }
        connections = PushRegistry.listConnections(true);
        if (connections == null || connections.length == 0)
        {
                output.append("Client waiting for messages.....",im);
        }
        done = false;
        thread = new Thread(this);
        thread.start();
        display.setCurrent(s);
}
public void notifyIncomingMessage(MessageConnection conn)
{
        if (thread == null)
        {
                done = false;
                thread = new Thread(this);
                thread.start();
        }
}
public void run()
{
        try
        {
                msg = smsconn.receive();
                if (msg != null)
                {
                        senderAddress = msg.getAddress();
                        if (msg instanceof TextMessage)
                        {
                                output.deleteAll();
                                output.append((TextMessage)msg).getPayloadText(),im);
                        }
                        else
                        {
                                StringBuffer buf = new StringBuffer();
                                byte[] data = ((BinaryMessage)msg).getPayloadData();
                                for (int i = 0; i < data.length; i++)
                                {
```

```
                                        int intData = (int)data[i] & 0xFF;
                                        if (intData < 0x10)
                                        {
                                                buf.append("0");
                                        }
                                        buf.append(Integer.toHexString(intData));
                                        buf.append(' ');
                                }
                                output.deleteAll();
                                output.append(buf.toString(),im);



                                }
                                display.setCurrent(s);
                        }
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
        }

        public void pauseApp()
        {
                done = true;
                thread = null;
                s = display.getCurrent();
        }
        public void destroyApp(boolean unconditional)
        {
                done = true;
                thread = null;
                if (smsconn != null)
                {
                        try
                        {
                                smsconn.close();
                        }
                        catch (IOException e)
                        {
                        // Ignore any errors on shutdown
                        }
                }
        }
        public void commandAction(Command c, Displayable s)
        {
                try
                {
                        if (c == exitCommand || c == Alert.DISMISS_COMMAND)
                        {
                                destroyApp(false);
                                notifyDestroyed();
                        }
                        else if (c == okCommand)
                        {
                                //Do nothing
                        }
                }
                catch (Exception ex)
                {
                        ex.printStackTrace();
                }
        }
        private void reply()
        {
        }
}
```

Business Layer Classes

TopicObserver

```
package com.mcebo.sanparks.mediator.notifiers;

import com.mcebo.sanparks.mediator.accesslayer.SubscriberDB;
import com.mcebo.sanparks.mediator.businesslayer.Folder;
import com.mcebo.sanparks.mediator.businesslayer.MailBox;
import com.mcebo.sanparks.mediator.businesslayer.Message;
import com.mcebo.sanparks.mediator.businesslayer.SubscriberProfile;
import com.mcebo.sanparks.mediator.businesslayer.Topic;
import java.util.ArrayList;
import java.util.Properties;
import java.util.*;

public class TopicObserver
{
        private Topic topic;
        private SubscriberDB sDB;
        private HashMap messageQueue;
        private String topicMessage;


        public TopicObserver(Topic observableTopic)
        {
                topic = observableTopic;
                sDB = new SubscriberDB("subscriberprofiles.xml");
                update();
        }
        public void update()
        {
             boolean isConnected = false;
             ArrayList subscribers = sDB.getSubscriberList(topic.getCategory());
             Object [] o  = subscribers.toArray();
                 for(int i = 0; i  < o.length;i++)
                 {
                    if(o[i] instanceof SubscriberProfile)
                     {
                        SubscriberProfile p =  (SubscriberProfile)o[i];
                        String deliveryMethod = p.getPreferences().getDeliveryMethod();
                        if(deliveryMethod.equals("SMS"))
                        {
                            try
                            {

                                Properties props = System.getProperties();
                                props.put("kvem.home","C:\\WTK23");
                                String topicMessage =  topic.getBody();
                                String destNumber = p.getMobileNumber();

MessageServer.sendMessage(topicMessage,destNumber.trim());


                            }
                            catch(Exception ex)
                            {
                                ex.printStackTrace();
                             }
                        }
                        else if(deliveryMethod.equals("Email"))
                        {
                                int mID = (int)(Math.random()*6000);
                                Message m = new Message();
                                m.setMessageID(mID);
                                m.setSenderAddress("topicsinfo@sanparks.co.za" ;
                                m.setDestinationAddress(p.getEmailAddress());
```

```
                                m.setReceivedDate(new Date().toString());
                                m.setMessageSubject(topic.getTitle());
                                m.setMessageContent(topic.getBody());
                                m.setStatus("unread");
                                Folder.saveMessage(m);


                            }
                            else if(deliveryMethod.equals("Both SMS and Email"))
                            {


                                Message m = new Message();
                                m.setSenderAddress("topicsinfo@sanparks.co.za");
                                m.setDestinationAddress(p.getEmailAddress());
                                m.setReceivedDate(new Date().toString());
                                m.setMessageSubject(topic.getTitle());
                                m.setMessageContent(topic.getBody());
                                m.setStatus("unread");
                                Folder.saveMessage(m);

                                String destAddress = p.getMobileNumber();

                              MessageServer.sendMessage(topicMessage,destAddress.trim());
                            }
                        }
                    }
        }
}
```

Profile

```
package com.mcebo.sanparks.mediator.businesslayer;

import java.io.Serializable;


public abstract class Profile implements Serializable
{


        protected String mobileNumber;
        protected String emailAddress;
        protected String password;
        public Profile(){}

        //setter methods
        public void setMobileNumber(String mobileNumber)
        {
                this.mobileNumber = mobileNumber;
        }
        public void setEmailAddress(String emailAddress)
        {
                this.emailAddress = emailAddress;
        }
        public void setPassword(String password)
        {
                this.password = password;
        }
        //getter methods
        public String getMobileNumber()
        {
                return mobileNumber;
        }
        public String getEmailAddress()
        {
                return emailAddress;
        }
        public String getPassword()
        {
```

```
                return password;
        }
}
```

PublisherProfile

```
package com.mcebo.sanparks.mediator.businesslayer;

import java.io.Serializable;

public class PublisherProfile extends Profile implements Serializable
{

        private String publisherID;
        public PublisherProfile(){}
        public void setPublisherID(String publisherID)
        {
                this.publisherID = publisherID;
        }
        public String getPublisherID()
        {
                return publisherID;
        }
}
```

SubscriberProfile

```
package com.mcebo.sanparks.mediator.businesslayer;

import java.io.Serializable;

public class SubscriberProfile extends Profile implements Serializable
{
        private String subscriberID;
        private Preferences preferences;

        public SubscriberProfile()
        {
                super();
        }
        public void setSubscriberID(String subscriberID)
        {
                this.subscriberID = subscriberID;
        }
        public String getSubscriberID()
        {
                return subscriberID;
        }
        public void setPreferences(Preferences preferences)
        {
                this.preferences = preferences;
        }
        public Preferences getPreferences()
        {
                return preferences;
        }
}
```

Preferences

```
package com.mcebo.sanparks.mediator.businesslayer;

import java.io.Serializable;

public class Preferences implements Serializable
{
```

```
        private String topicCategory;
        private String deliveryMethod;
        private String keyWord;
        private String parkName;
        public Preferences(){}

        //Setter methods
        public void setTopicCategory(String topicCategory)
        {
                this.topicCategory = topicCategory;
        }
        public void setDeliveryMethod(String deliveryMethod)
        {
                this.deliveryMethod = deliveryMethod;
        }
        public void setKeyWord(String keyWord)
        {
                this.keyWord = keyWord;
        }
        public void setParkName(String parkName)
        {
                this.parkName = parkName;
        }

        //Getter methods
        public String getTopicCategory()
        {
                return topicCategory;
        }
        public String getDeliveryMethod()
        {
                return deliveryMethod;
        }
        public String getKeyWord()
        {
                return keyWord;
        }
        public String getParkName()
        {
                return parkName;
        }
}
```

Topic

```
package com.mcebo.sanparks.mediator.businesslayer;

import java.io.Serializable;


public class Topic implements Serializable
{
        private String title;
        private String category;
        private String body;
        private String keyWord;
        public Topic()
        {
        }
        //Setter methods
        public void setTitle(String title)
        {
                this.title = title;

        }
        public void setCategory(String category)
        {
                this.category = category;
```

```
        }
        public void setBody(String body)
        {
                this.body = body;

        }
        public void setKeyWord(String keyWord)
        {
                this.keyWord = keyWord;

        }

        //Getter methods
        public String getTitle()
        {
                return title;
        }
        public String getCategory()
        {
                return category;
        }
        public String getBody()
        {
                return body;
        }
        public String getKeyWord()
        {
                return keyWord;
        }
}
```

Message

```
package com.mcebo.sanparks.mediator.businesslayer;

public class Message
{
        private int messageID;
        private String senderAddress;
        private String destinationAddress;
        private String messageSubject;
        private String messageContent;
        private String receivedDate;
        private String status;
        public Message(){}
        public String getSenderAddress()
        {
            return senderAddress;
        }
        public void setSenderAddress(String senderAddress)
        {
            this.senderAddress = senderAddress;
        }
        public String getDestinationAddress()
        {
            return destinationAddress;
        }
        public void setDestinationAddress(String destinationAddress)
        {
            this.destinationAddress = destinationAddress;
        }
        public String getMessageSubject()
        {
            return messageSubject;
        }
        public void setMessageSubject(String messageSubject)
        {
            this.messageSubject = messageSubject;
```

```
        }
        public String getMessageContent()
        {
            return messageContent;
        }
        public void setMessageContent(String messageContent)
        {
            this.messageContent = messageContent;
        }
        public String getReceivedDate()
        {
            return receivedDate;
        }
        public void setReceivedDate(String receivedDate)
        {
            this.receivedDate = receivedDate;
        }
        public String getStatus()
        {
            return status;
        }
        public void setStatus(String status)
        {
            this.status = status;
        }

    public int getMessageID()
    {
        return messageID;
    }

    public void setMessageID(int messageID)
    {
        this.messageID = messageID;
    }

}
```

Folder

```
package com.mcebo.sanparks.mediator.businesslayer;

import com.mcebo.sanparks.mediator.accesslayer.MessageDB;
import java.util.ArrayList;

public class Folder
{
    private String folderName;

    //Counting messages
    private int totalMessageCount;
    private int newMessageCount;
    private int  readMessageCount;

   //Retrieving messages
    private Message []  totalMessages;
    private static MessageDB  mailDB;
    public Folder()
    {

    }
    static
    {
            mailDB = new MessageDB("mails.xml");
    }

    public String getFolderName()
    {
```

```
        return folderName;
    }
    public void setFolderName(String folderName)
    {
        this.folderName = folderName;
    }

    //Getting number of new messages
    public void setNewMessageCount(int newMessageCount)
    {
        this.newMessageCount = newMessageCount;
    }
    public int getNewMessageCount()
    {
          return newMessageCount;
    }

    //Getting the number of read messages
    public void setReadMessageCount(int readMessageCount)
    {
            this.readMessageCount = readMessageCount;
    }
    public int getReadMessageCount()
    {
            return readMessageCount;
    }

    //Getting the number of total messages
    public void setTotalMessageCount(int totalMessageCount)
    {
        this.totalMessageCount = totalMessageCount;
    }
    public int getTotalMessageCount()
    {
        return totalMessageCount;
    }
    public void  markAsReadMessages(int messageID)
    {
            mailDB.markAsReadMessages(messageID);
    }
  //Getting all new messages
    public Message[] getMessages(String emailAddress)
    {
            int read = 0;
            int unread = 0;
            totalMessages =  mailDB.retrieveMessages(emailAddress);
            int len = totalMessages.length;
            for(int i = 0;i < len;i++)
            {
                    if(totalMessages[i].getStatus().equals("unread"))
                        unread++;
                    if(totalMessages[i].getStatus().equals("read"))
                        read++;
            }
            setNewMessageCount(unread);
            setReadMessageCount(read);
            setTotalMessageCount(len);
            return totalMessages;
    }

    public static void saveMessage(Message message)
    {
           mailDB.storeMessages(message);
           System.out.println("Saved");
    }
    public void deleteMessages(int messageID)
    {
         mailDB.deleteMessageRecord(messageID);
    }
}
```

APPENDIX D

USABILITY TESTING INSTRUMENTS

Questionnaires for the Usability testing of the new SANPARKS Information System.

Performance Evaluation of the new SANPARKS Information System!!

We base the evaluation of the new system on usability testing, since the aim of this research is to improve the users' friendliness of the system. In order to achieve that goal, there is the need to interact with users by means of instrument (Questionnaires) to capture the view of the users about the new system.

SECTION A – People from IT-related Discipline

System development is the result of applying programming languages and software related courses to achieve the solution of the defined problem in a particular study. People who are much involved in systems development are Computer Science, Information Technology and Library Information Systems students. We will then interview some students from the above mentioned departments in order to check whether restructured SANPARKS system meets the following design criteria: Restructuring, Transparent Information Delivery and Personalization. Below is the full meaning of the design criteria's.

1. Restructuring - Make the SANPARK system fit for the status of information provider.

2. Transparent Information Delivery - Integrate a publish/subscribe engine that allows a consumer to get information without knowing the source.
3. Personalization – Enhance the engine to serve mobile users in a personalized manner.

The students suppose to take a tour on the system and use the system by subscribing to the system and wait for the information to be delivered to them by the publishing students. When the information is already delivered, students need to check whether the information meets their interest. Students suppose to divide themselves according to two categories as this system is the publish/subscribe system. The must be students that will play a role of subscribers and some will be publishers. In completion students needs to answer the following questions.

Please note: The questionnaire is in two phase (Subscriber/Publisher).

Instructions: Shade circles or simple mark X in response.

Subscribers (Students) Questionnaires

1. Please also tick your department from the list below.

O Computer Science

O Library Information System

O Information Technology

2. Which level of study are you doing?

O First year

O Second year

O Final year

O Honors

O Masters

O Doctorate

3. How is the restructured SANPARKS system respond in terms of the following?

Parameters	Good	Fair	Excellent
1. Quality of Interaction			
2. Quality of Information Provided			

4. How do you find the SANPARKS system when taking a tour on it (using it)?

O Enjoyable

O Boring

O Satisfactory

5. In your opinion is this system user friendly (or easy to use)?

O Yes

O No very complicated

O Average, normal as other system

6. Does this SANPARKS system satisfy your interest of the services?

O Yes

O No

7. Does this system support personalization of services when it delivers notifications to you as a client?

O Yes

O Not at all

8. Do you think information provided by this system is limited and requires some more improvements?

O Yes

O No, if yes specifies what need to be improved.

.....

.....

.....

.....

Publishers (Students) Questionnaires

1. Please also tick your department from the list below.

O Computer Science

O Library Information System

O Information Technology

4. Which level of study are you doing?

O First year

O Second year

O Final year

O Honors

O Masters

O Doctorate

5. Is this SANPARKS system fit for the information provider entity?

O Yes

O No

6. Do you think the new SANPARKS system able to reach correct prospective clients?

O Yes

O No