Context-Aware Advertising in Wireless Ad-Hoc Networks

Colm Caffrey

A dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science

2007

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Colm Caffrey

September 14th, 2007

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Colm Caffrey

September 14th, 2007

Acknowledgements

I would like to thank my supervisor, Dr René Meier for all his help, guidance and enthusiasm throughout the course of this dissertation. I would also like to thank my friends, family and particularly Sarah for her help and encouragement throughout.

COLM CAFFREY

University of Dublin, Trinity College

2007

Abstract

There has been much recent research into the area of pervasive computing and the development of collaborative applications. Increased levels of mobility in computing devices combined with the ability to dynamically create mobile ad-hoc networks offers great potential for the development of 'smart' applications. Application nodes can spontaneously interact and communicate with each other without a reliance on any infrastructure. In this type of environment an application node's contextual information is likely to be similarly dynamic and subject to frequent change. This type of environment is therefore ideally suited to creating highly adaptive applications which can react and tailor their behaviour to their current context.

One application domain which could benefit from this type of application is that of mobile advertising. There is an increasing popularity of information services that provide various types of content delivery in mobile environments. As consumer usage has shifted more and more from traditional media towards mobile and wireless devices, advertisers have inevitably begun to shift their focus towards these new forms of media. However, the intimate and personal nature of mobile devices means that users expect more personalised and intimate content on these devices. In order for mobile advertising to be successful and to gain widespread consumer acceptance, advertising content delivered to users must be highly personalised and relevant.

This dissertation investigates the potential for creating a context-aware advertising application deployable in a mobile ad-hoc environment. The dissertation builds on existing research to develop mobile event-based middleware and applies it to the domain of advertising. The research includes the design and implementation of an advertising system using this existing middleware. The system enables consumers moving into an

advertisement space to receive advertisements from service providers based on their context. Advertisers and consumers communicate in an ad-hoc manner by dynamically creating communication links with other application nodes within close proximity. Nodes are free to enter and exit the application space, without affecting the operation of the application. By creating a collaborative application environment, advertisers can benefit by delivering more targeted and personalised advertisements to consumers, while consumers experiences are improved by receiving more relevant and personalised advertisements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advertising can be defined as the act of presenting or describing a product, service, or event in a public medium [3]. Traditionally advertisements were designed for consumption by multiple consumers and were therefore created in a non-personal form, suitable for consumption by the masses. Advertisements have become more widespread and ever-present in our daily lives and interactions, to the point where they are now almost unavoidable and dominate all forms of public media. Successfully gaining an individual's attention and delivering an advertisement message to them has become a major challenge and intensely competitive among advertisers. As a result, advertisers have had to become increasingly innovative in their approaches.

Advertising as a discrete form is widely agreed to have begun with newspapers in the seventeenth century. Since its conception, advertising has evolved and responded to changing business demands, media technologies, and cultural contexts. As new forms of mass media have emerged, advertising inevitably also appears on these new media. The introduction of cinema, radio and television to consumers was quickly proceeded by the presence of advertising on these media, as advertisers attempted to leverage their potential for reaching mass consumer markets. Similarly, the relatively new area of online advertising has seen rapid growth as the web has become a predominant part of our daily lives. The interactive nature of the web and personal computing has created a new set of possibilities for advertisers to deliver more personalised messages than was previously possible with traditional media. Advertisements are therefore evolv-

ing from their traditional 'non-personal' nature to more user-specific and personal messages. This rapid growth of online advertising is evident in recent statistics published that show it has already surpassed newspaper advertising and currently accounts for 11.4% of all advertising revenue in the UK [4].

The increasing pervasiveness of mobile devices has provided advertisers with another potential media outlet to deliver their messages to consumers. Indeed, advertisers are already investing heavily in this area. According to eMarketer [5], mobile marketing and advertising in the US is expected to grow from US$421m to US$4.8bn from the period of 2006 to 2011, accounting for 12pc of overall online advertising spending. This indicates the potential for innovative technologies and applications in this growing area. The personal and intimate nature of mobile devices offers potential for advertisers to deliver advertisement messages that are more personalised and targeted at a user's specific interests.

An important part of the growing area of mobile and ubiquitous computing, deals with the idea of context-awareness and building applications that can: acquire context, understand or interpret the acquired context, and modify or adjust an applications behaviour based on this context. Dey [6] defines context as "any information that can be used to characterise the situation of entities". The acquisition and interpretation of contextual information (e.g. location) in the domain of mobile advertising applications provides information that could potentially be leveraged in order to deliver more adaptive and targeted advertisements than is currently possible. In doing so, advertisers could increase their penetration of interested consumers, while consumers' experiences could be improved by increasing the relevance of any advertisements delivered to them.

1.1 Background and Motivation

Advertising has grown phenomenally in recent years and has become a dominant and unavoidable presence in our daily lives. Consumers are exposed to hundreds and sometimes even thousands of advertisement messages per day in one form or another. As a result many consumers now actively take steps to avoid being 'disturbed' by advertisements. Indeed, there is some evidence to suggest that advertising is almost reaching saturation point. Recent marketing research published [4] indicates that there has

been a significant reduction in the growth of online advertising as growth has fallen from 30% for the last three years to 19% for 2007. While online advertising is still growing, the rate of this growth is slowing. This trend appears to be part of an overall advertising slowdown across all media and is possibly indicative of increasing consumer resistance to advertising.

As consumer media usage continues to shift from traditional media to computer and mobile technologies, advertisers' focus has and will continue to shift to these forms of media. The area of mobile advertising in particular is a new and evolving one. Mobile computing and also context-aware computing present many new and interesting possibilities in terms of meeting the challenges being faced by advertisers to deliver more personalised and relevant advertisements. Successfully providing this level of relevance will be key to user acceptance in this emerging area.

Recent advances in wireless technology and mobile computing has enabled the creation and emergence of self-organising, rapidly deployable network architectures referred to as 'ad-hoc' networks. An ad-hoc network is a self-configuring network of wireless mobile nodes which dynamically form a temporary network. Nodes in the network are free to move randomly and re-organise, without the aid of any existing infrastructure or centralised configuration or administration services. Multi-hop routes are dynamically determined in order to transmit messages between nodes that are not within direct transmission range of each other [7]. The dynamic and spontaneous nature of ad-hoc networks provides potential in many varying scenarios to create collaborative applications. In effect a network can be formed with anybody, anytime and anywhere. This provides great flexibility, particularly for application scenarios where having prior knowledge of application nodes, or providing infrastructure at all potential locations, is not feasible. One such application scenario is in the domain of advertising. Advertisers typically do not have prior knowledge of the potential consumers of their advertisements. Communication with consumers is usually temporary, only lasting long enough to deliver their advertising message to the consumer. Consumer locations are also variable and may cover a large geographic area. A collaborative ad-hoc application environment would allow advertisers to dynamically communicate and deliver their message to potential consumers. The type of mobile advertising application scenario possible is analogous to the traditional advertising method of distributing promotional flyers: an advertiser typically wanders within an area in a random fashion. As they identify and encounter

potential consumers, they initiate communication with them in a spontaneous manner and deliver their advertising flyer to them. The advertiser continues moving and looking for more potential consumers and the consumer may subsequently pass on the flyer to some other consumer it encounters. Ad-hoc communication technology could help to enable this type of traditional advertising scenario to be replicated in a mobile computing application environment.

The publish/subscribe paradigm is a well known asynchronous messaging paradigm that enables message senders (publishers) to publish messages without having any prior knowledge of the consumers of the messages. This enables network entities to operate in a loosely-coupled and autonomous fashion and also helps to provide operational immunity from network failures. Various middleware products have been developed in order to facilitate this type of communication model and have been generally characterised under the message oriented middleware (MOM) category of middleware. In the publish/subscribe application scenario, publishers publish messages without having any prior knowledge of the message's recipients. Consumers of messages do not specify the publisher to receive events from, but instead subscribe or register an interest in receiving messages of a particular type. They subsequently only receive messages of this type. This paradigm is again a natural fit to the scenario of an advertising application where advertisers do not typically have any prior knowledge of the identity of the consumers of the advertisement.

Context-aware computing is a mobile computing paradigm in which applications can discover or sense context and subsequently react to or take advantage of this acquired contextual information [8]. Context, in terms of mobile computing is defined in [9] as "information that is of an application's environment and that can be sensed by the application". One example of a type of context that has been commonly used to date is geographic location. Applications utilising geographic location information are said to be 'location-aware' and can provide services that fit a user's current location. By incorporating location and other forms of context into applications, it is possible to create systems that continuously adapt to a user's environment. As a result, applications can potentially be more effective and provide a more pleasant experience to the end-user than was previously possible.

1.2 Goals and Objectives

The overall goal of this dissertation is to investigate the potential for advertising applications in this type of context-aware, ad-hoc environment. It is intended to create a collaborative application environment which increases the relevance and personalisation of advertisements delivered to consumers and provides advertisers with improved targeting capabilities. The intention is to build on previous research that has been conducted to develop a middleware enabling the development of collaborative applications in ad-hoc environments [2, 10]. This research aims to extend this existing middleware to create an additional domain-specific layer suitable for developing advertising applications, and subsequently develop an application that utilises this additional layer. The main goals and objectives can therefore be summarised as follows:

- Evaluate the current state of the art in the key relevant areas.
- Investigate the potential for advertising applications in mobile ad-hoc collaborative environments.
- Extend existing middleware to cater specifically for the development of advertising applications in collaborative environments.
- Evaluate the system to determine its overall value and potential.

1.3 Dissertation Outline

The structure of the remainder of this dissertation document is outlined as follows:

Chapter 2: examines and reviews the current state of the art in the main areas relevant to the dissertation.

Chapter 3: outlines the system design, in terms of high level design goals and objectives and a detailed discussion on design decisions made.

Chapter 4: outlines the implementation of the design as outlined in Chapter 3, including key issues encountered during this phase.

Chapter 5: comprises an evaluation and analysis of the dissertation.

Chapter 6: presents a conclusion, including a summary of the achievements and identifies possible areas of improvement and future work.

Chapter 2

State of The Art

This chapter reviews the current state of the art in the key areas relevant to this research, namely context-awareness, mobile advertising, ad-hoc networking and event-based middleware. Each section reviews research in its specific area that has been deemed relevant to the scope of this dissertation.

2.1 Context-Awareness

Context-aware computing refers to a computing paradigm in which applications can discover and take advantage of contextual information. The behaviour of an application is therefore determined by the environment or circumstances in which it finds itself. It can be broadly described as an entity's ”situational information”. There have been numerous attempts to concisely define context in terms of mobile computing but as of yet, there is no single agreed definition. One proposed definition [11], defines context as *"any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves"*. Contextual information can be viewed as being transient or persistent [12]. Transient refers to information at a particular moment in time whereas persistent refers to a history of transient context.

Contextual information can therefore refer to almost any information that is available to an application at a particular moment in time. Some examples of information that

can be considered as contextual are identified in [13]: identity; spatial information (e.g. location, orientation); temporal information (e.g. time of day); environmental information (e.g. temperature, noise level); social situation; resources that are nearby; availability of resources (e.g. battery, bandwidth); physiological measurements; activity schedules and agendas. A system or application can be said to be context-aware if it can extract, interpret and use context information and subsequently adapt its behaviour to the current context of use. The challenge for such context-aware systems, lies in the complexity of capturing, representing and processing the acquired contextual data.

Moore and Hu [14] suggest that context can fall into one of two general classifications:

- Static Context (customisation): a state in which the 'look-and-feel' and content provision is user-driven and the user has an element of control.
- Dynamic context (personalisation): a state in which the user is passive, or at least somewhat less in control. The system reacts to the users situated role, location, actions and behaviour.

The paper describes two principal ways in which context may be used. The first is the encoding of context to enable its use as a 'retrieval clue', while the second is its use to dynamically tailor system behaviour to usage patterns. It is suggested that a combination of both of these approaches should be used in order to produce an overall context definition.

There has been much recent growth in the use and availability of wireless and mobile devices. These types of devices by their nature lead to more dynamic context. The operating environment of such devices is constantly changing due to their mobility and the characteristics of wireless communication technologies. Sensor technologies (e.g. GPS) have also become increasingly cheap, small, and accurate. These combining factors has lead to even greater potential for creating highly adaptive and context-aware applications.

Another definition for context-aware computing is provided by Barkhuus and Dey [15] as *"an application's ability to detect and react to environment variables"*. It is proposed that the level of interactivity between a mobile computing device and its user can be categorised into one of three levels:

- Personalisation: the user specifies their own settings for how the application should behave in a given situation. It can also be referred to as customisation or tailoring.
- Passive context-awareness: presents updated context or sensor information to the user but lets the user decide how to change the application behaviour.
- Active context-awareness: autonomously changes the application behaviour according to the sensed information.

The research aims to test which approach will limit users' perceived sense of control and to investigate users' preferences towards the different approaches. The results of the research conclude that with passive and active context-awareness, users felt less in control. However, despite this perceived lack of control, users preferred the context aware applications. It seems therefore that once an acceptable level of control is determined, at which a user is still prepared to use a system, then the user can enjoy an improved experience provided by a system's context-awareness.

Drogehorn et al. [16] discuss personalisation as a means of easing the use of pervasive computing systems. In general it is found to improve and increase the value of a user's experience. Profiles are identified as one of the key building blocks required to make personalisation happen. They discuss the use of context and personalisation in pervasive systems. Context of the user enables services to predict the users intention and context-awareness is meant to capture the situation of the moment. Profiles provide configured and learned behaviour, preferences and capabilities of users, devices, networks and services to a personalisation system.

Cyberguide [17] is a mobile context-aware tour guide and is an example of one of the earlier context-aware applications that has been developed. It is a location-aware tour guide in which the user's location and orientation is the contextual information acquired. This information is then used to present relevant tour guide information to the user. Applications such as this have proved to be useful in their targeted environments. However, one major feature of this and other similar applications is the reliance upon prior knowledge of the parties with which they interact and a reliance on communications and/or sensor infrastructures. Many newer forms of context-aware applications are focused on ad-hoc communications and rely on opportunistic and spontaneous interactions. A different approach is therefore required in order to meet the challenges faced in developing applications capable of operating in this type of ad-hoc environment.

2.2 Mobile Advertising

While mobile advertising is certainly a growing area, it has become apparent that in order for mobile advertising to become successful, the notion of 'targeted' advertising needs to be addressed for mobile environments. Mobile advertisements must be highly personalised and go beyond basic demographic profiling that may be used in traditional forms. Due to the very personal nature of mobile devices, it is important to only deliver advertising that is strictly related to personal interests. Advertisement 'relevance' needs therefore to be redefined for mobile environments.

Aalto et al. [18] present a system for delivering permission-based location-aware advertisements to mobile phones. The system uses bluetooth positioning and wireless application protocol (WAP) push. They discuss the large potential of mobile advertising due to the intimate nature of mobile devices and high targeting possibilities. Their research showed that bluetooth-based positioning was not very reliable or real-time and in many cases users were not detected, leading to a failure for advertisements to be triggered. This is not however a major issue for the users of an advertisement service as they typically don't know when to expect an advertisement. The authors also identify what they consider to be the key desirable features of any mobile advertising system. These features are summarised as follows:

- Should be augmented with profiling or personalisation in order to ensure that only relevant, targeted advertisements are sent or offered to users.
- Should be possible to limit the frequency of the advertisements.
- Profiling should be set up when a user subscribes to the advertisement service.
- Possible profiling factors identified are: gender, age, language, interests, mood, advertising frequency.
- A system with the ability to learn user preferences is also identified as being of high potential.

Bulander et al [19] discuss the potential and challenges of mobile advertising while presenting the MoMa-system for delivering personalised and context-sensitive advertising.

Benefits or positive aspects identified of mobile advertising include: high penetration rate of mobile devices, personal nature of mobile devices, the fact that the devices are individually addressable, multimedia capabilities and interactive nature of mobile devices. Potential problems or challenges identified are as follows: potential for spam abuse; limited user interfaces of mobile devices; consumer privacy concerns; expense of mobile communications. A key focus of the MoMa system is to guarantee data protection due to the privacy concerns with mobile advertising. The system also distinguishes between public and private context.

Hristova and O'Hare [8] present a mobile advertising system named Ad-Me. Ad-Me is a context-sensitive advertising system, its aim being to deliver personalised and unintrusive advertisements to mobile devices. It also discusses the possibility of using user emotion to increase effectiveness of mobile advertisements. Advantages identified of wireless advertising include the fact that it is: cost-effective, personal, and has potential for location-aware and personalised advertisements unlike traditional media. Some potential security threats are also identified for wireless advertising such as: the fact that network decision making is decentralised; the potential for distributing misinformation; malicious downloads; denial of service attacks; collection of personal preferences and geographic location data. Some limitations of wireless advertising also arise from device limitations such as: limited memory, network bandwidth and screen real estate. Efficiency in advertisement delivery in this type of mobile scenario is therefore key. Ad-Me uses passive and active user profiling i.e. passive at registration time and dynamic as the user uses the system. A number of key issues are identified that should be considered in the development of mobile/wireless advertising applications:

- Context definition and utilisation
- User and advertiser profiling
- Incorporating geographical information
- Device profile considerations
- Effect/benefit for sales revenue

"eNcentive: a framework for intelligent marketing in mobile peer-to-peer environments" is presented by Ratsimor et al. [20]. The framework facilitates peer-to-peer electronic

marketing in mobile ad-hoc environments. It employs an intelligent marketing scheme by collecting information like sales promotions and discounts, and marketing it to other users in the network. As a user passes by businesses offering promotions, the eNcentive platform running on the user's PDA actively collects and caches these coupons and promotions. A user can also employ the eNcentive platform to become a distributor of these coupons and advertisements. In this case, the platform starts to actively advertise coupons to other eNcentive peer platforms that the user passes by along the way. Every coupon contains a list of platform IDs of every eNcentive platform that ever distributed that particular coupon. This enables participating businesses to reward their most effective distributors.

Podnar et al. [21] analyse the features of a mobile push service and propose an architecture for mobile content delivery systems. The architecture proposed supports many to many interactions and is based on the publish/subscribe paradigm. They argue that in order for location-based content delivery systems to be widely accepted, any such system must be capable of delivering highly personalised and customised content. A publish/-subscribe architecture is ideally suited to mobile environments where devices may be frequently unavailable or disconnected. This is due to the decoupling and asynchronous interactions between nodes provided by the architecture. A number of additional requirements introduced by mobile environments are identified: the routing problem is more complex; system needs to be resilient to frequent disconnections; requirement to handle duplicate messages. The publish/subscribe infrastructure must address these mobility issues explicitly. A four-layer architecture for mobile push systems is proposed, comprised of the following layers:

- Transport layer: utilises TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).
- Communication layer: enables the interaction between publishers and subscribers and has a distributed architecture to address scalability and implements a routing algorithm that supports user mobility.
- Service layer: contains the utility services needed by mobile push such as location management, adaption management, user profile management, subscription management.

- Application layer: push-specific layer that co-ordinates other services, manages and stores device dependent content, manages the transfer of information between content dispatcher.

A recent study into mobile advertising conducted by Harris Interactive [22] looks into consumer acceptance of mobile phone advertisements in the US. The research examines current levels of consumer interest in mobile phone advertisements, preferred advertising formats and also the willingness of consumers to be profiled. Historically, US mobile phone users have been resistant to receiving mobile phone advertisements. According to the research though, mobile phone users are now more willing than ever to receive advertising. The survey reveals that mobile phone users are most interested in receiving advertisements that have a clear value proposition, are relevant, and allow recipients to control how they are profiled. Users who are at least somewhat interested in mobile advertising identified the following as incentives to 'lessen the pain' in receiving these adverts:

- The ability to 'opt out'.
- Ability to choose the type of ads to be received.
- Ability to choose the number of advertisements to be received in a given period of time.
- Providing a profile of desired areas of interest so only specific ads are sent.
- Inclusion of discounts or other incentives with advertisements.
- Ability to choose specific times when advertisements should be received.

2.3 Ad-Hoc Networks and Collaborative Applications

Ad-hoc networks enable wireless mobile nodes to dynamically form a network without the need for an existing infrastructure. They are characterised by being self-organising and rapidly deployable network architectures. These characteristics offer much potential in terms of the types of dynamic and collaborative application environments that

can be created. However in addition to this great potential, a number of challenges exist in order to facilitate the creation of such applications. For example, there is great diversity in the numbers of currently available devices, networking infrastructure and information content for such applications. Many applications have tended to provide specialised solutions, only applicable to their specific domain. It is important to attempt to provide mechanisms that weave together the multitude of sensing, computing, communication and information technologies available. Recently there has been a large focus on research in this area, to provide middleware to ease application development and simplify the creation of ad-hoc communities for mobile applications.

One such middleware is presented in EMMA (Epidemic Messaging Middleware for Ad hoc networks) [23]. This middleware is an adaptation of the Java Message Service (JMS) and enables it for mobile ad-hoc environments. The paper examines an epidemic routing mechanism that enables delivery of messages in a partially connected network to attempt to demonstrate the feasibility of the solution. They suggest that middleware for mobile ad-hoc environments presents new and fundamental questions to researchers. It is necessary to investigate and devise novel architectural patterns to answer these fundamental questions, which cannot be solved by applying principles and patterns traditionally exploited in the design of middleware for fixed systems.

2.4 Event-Based Middleware

Message-oriented and event-based middleware refers to types of middleware that rely on asynchronous message-passing rather than the traditional request/response style of communication. It enables many-to-many communication in an asynchronous manner where senders and receivers of messages are loosely coupled. This provides increased scalability, particularly for large-scale distributed systems. Message-oriented middleware generally uses a message queue system or alternatively rely on a broadcast or multi-cast messaging system. The abstractions provided by such middleware hide the complexities of the network underlying communication protocols, enabling application developers to more easily develop distributed applications.

The publish/subscribe paradigm is a well known paradigm that is commonly used to provide this type of asynchronous communication between message senders and receivers.

Senders do not require knowledge of the receivers of the message but instead describes the characteristics of the message by assigning a message 'type'. Interested parties can then subscribe to this event type in order to receive this and other messages of this type. The publish/subscribe paradigm has been extended to operate in distributed mobile environments. This enables systems to cope with dynamically changing environments in which publishers and subscribers can become disconnected from the network without affecting the overall operation of the system.

Huang and Garcia-Molina [24] discuss the extension of the publish/subscribe paradigm to distributed environments. They discuss various types of distributed architectures and the potential issues with adapting publish/subscribe communication to these architecture. In particular, they focus on mobile and wireless environments. They propose a solution that relies on a system of Event Sources, Event Brokers and Event Displayers. This solution requires centralised Event Brokers or a distributed version of these Event Brokers. There is a level of reliance therefore on an infrastructure of these event brokers in order to create applications that utilise this type of framework.

Podnar et al. [21] propose another architecture for mobile content delivery systems based on the publish/subscribe paradigm. The aim is to address some of the inherent issues associated with applying the publish/subscribe paradigm to mobile environments and to in effect design an architecture to enable mobile push based systems. The solution supports nomadic and mobile application nodes but also relies upon a network infrastructure in order to achieve network communication.

2.4.1 Jini

Jini can be described as a set of specifications that enables services to discover each other on a network [25]. It provides a framework that allows services on a network to participate in various operations. Jini blurs the distinction between hardware and software on a network by treating all entities as services. Its aim is to allow service interaction to occur in a dynamic and robust way where consumers of services don't require prior knowledge of a service's implementation. It is therefore ideal for creating dynamic environments in which mobile components come and go. The term Jini is used to refer to both a specification and an implementation. Both were initially created by

Sun Microsystems and have since been released under the Apache 2.0 license and offered to the Apache Software Foundation's Incubator project. The Jini project website [26] defines it as follows:

> Jini technology is a service oriented architecture that defines a programming model which both exploits and extends Java technology to enable the construction of secure, distributed systems consisting of federations of well-behaved network services and clients.

A Jini system is composed of a number of parts. The components are distinct but are also interrelated:

- A set of components that provides an infrastructure for federating services in a distributed system.
- A programming model that supports and encourages the production of reliable distributed services.
- Services that can be made part of a federated Jini system and that offer functionality to any other member of the federation

The goals of a Jini system include: enabling users to share services and resources over a network; providing users with easy access to resources on the network while allowing a potentially dynamically changing user network location; simplifying building, maintaining, and altering a network of devices, software, and users. Jini in effect extends the Java application environment from a single virtual machine to a network of machines. The Java application environment is naturally suited to distributed computing. It allows for code mobility, where both code and data can move to different nodes. Its built-in security also means that code downloaded from another machine can be run in confidence.

The Jini architecture extends this code mobility and adds mechanisms that allow fluidity of all components in a distributed system. The Jini technology infrastructure provides mechanisms for devices, services, and users to join and detach from a network.

Programming Model

The Jini infrastructure both enables the programming model and makes use of it. Entries in the lookup service are leased, allowing the lookup service to reflect accurately the set of currently available services [27]. The Jini programming model includes three facilities to help solve the problems commonly encountered in distributed computing: leasing, distributed events, and transactions.

Leasing is used to allocate resources in a distributed environment. The leasing interface adds time to the notion of holding a reference to a resource. This allows for recovery after network failures by enabling references to be reclaimed safely. Leases can be cancelled, renewed, allowed to expire, or manipulated by a third party.

The event and notification interfaces extend the standard Java event model for use in distributed systems. It enables events to be handled by third-party objects while making various guarantees regarding delivery and timeliness. It also recognises and provides mechanisms for dealing with the fact that the delivery of a distributed notification may be delayed.

The transaction interfaces introduce a lightweight protocol enabling Jini applications to coordinate state changes, involving a 'voting' and 'commit' phase.

Distributed Events

Jini extends Java's standard events mechanism by defining a set of interfaces and conventions for distributed events in the Jini Distributed Event Specification [28]. Distributed events are more complex than local events due to their requirement to deal with issues such as potential network failures. In contrast to local events, distributed events may potentially be: delivered out of order; not delivered; delivered slowly. The Jini Distributed Events model defines a set of interfaces and conventions that allow objects to identify, register for, and send notifications of events between different Java Virtual Machines which may be on different hosts.

As part of the distributed event model, Jini also enables third parties to provide software components, called adaptors. These adaptors can be positioned between the producer (event generator) and the consumer (event listener). This allows for the event service to be extended in order to provide different features such as delivery guarantees, filtering or buffering of events.

2.4.2 STEAM

STEAM (Scalable Timed Events And Mobility) is approach to event-based middleware for mobile computing presented by Meier and Cahill [1, 10]. Similar work in the area prior to this research focused on applications whose communication relied on fixed communications infrastructure. STEAM on the other hand is focused on collaborative applications that employ dynamically created ad-hoc networks as the basis for network communication. It is therefore intended to be suitable for creating collaborative applications that can create smart environments by dynamically creating connections to other application nodes within close geographic proximity of each other. Its lack of reliance upon any type of pre-existing infrastructure is considered one of its major strengths. It also provides location-awareness. A location service uses sensor data to provide geographic location information to the application. This data is used to restrict event delivery to circumstances where a node is at a location that is considered relevant. In addition to this the middleware provides distributed event filters, applicable at both consumer and producer sides and which can be applied to functional or non-functional requirements.

There are four key features central to the design of the STEAM architecture which are summarised as follows:

1. Supports collaborative application components which can come together at a different locations and dynamically create ad-hoc networks using wireless communications.
2. Inherently distributed architecture: the middleware resides on the same machine as the application and does not therefore rely on any designated infrastructure.
3. Supports a range of event notification filters that may be applied to various attributes of event notifications, including subject, context, and content.
4. Scalable and capable of coping with a large and dynamically changing population of mobile components which are potentially distributed over a large geographic area.

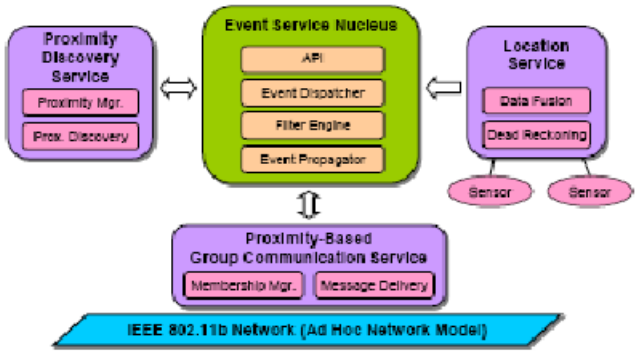

Figure 2.1: STEAM Architecture [1]

Figure 2.1 illustrates the architecture design of the STEAM middleware. There are four distinct components or services that comprise the architecture: location service; proximity discovery service; Proximity-Based Group Communication Service; Event Service Nucleus. Key features of the middleware are its ability to allow a producer to publish events of a specific 'event type' and to bound its range of propagation to a defined proximity range. The defined proximity range is independent of a nodes physical transmission range. This is achieved by using multi-hop to propagate a message between nodes within an ad-hoc network. Proximity can be defined as stationary or mobile. A mobile proximity will move as the node that defines it moves whereas a stationary proximity will remain at the position it was defined and not change as the producer changes location.

Proximity is defined according to a geometric shape, the shape and dimensions of which are entirely flexible and can be defined by the producer according their specific application requirements. Proximities may also be defined as nested and overlapping areas. Nodes residing within a proximity that overlaps with another proximity of a different event type will not interfere with propagation of these other event types.

Three event notification filters are central to the STEAM architecture: subject, content, and proximity filters. The combination of these filters can be used in order to filter any received events. Producers disseminate announcement and event messages. Announcement indicate the types of events they intend to send and the event message contains the actual message content. A consumer must subscribe to a particular event type before any events of that type are delivered to the application. Upon receiving an

announcement, a consumer makes a decision whether or not to subscribe to the contained event type. Any subscriptions remain persistent between multiple proximities and therefore a single subscription potentially enables a consumer to receive multiple events of this type in multiple proximities and from multiple producers.

Communication is based on a group communication transport mechanism and employing a multicast protocol for message routing. In order to apply for group membership, an application node must be within the geographical area of the group and be interested in joining the group. Unique identifiers are generated using subject and proximity pairs. This means that when a node receives a message, it will either recognise its identifier and deliver it, or otherwise not recognise it and discard it. The use of multicast messaging leads to increased scalability. The message sender's resource usage does not increase as the number of consumers increases.

Every mobile device maintains a repository containing the filters that describe the proximities that are available at the device's current location. It also contains type information describing the events that are associated with these proximities.

The routing approach involves routing multi-hop messages in multicast groups. As a result, an application node will only forward a message if it is of an event type that it is interested in. This is the case for both announcement messages and event messages. However, all devices located within a proximity effectively forward announcement messages even if they are not interested in the contained event type and proximity. This is attributed to the fact that all devices need to discover proximities and use the well-known discovery group when doing so.

2.4.3 PENCA

Related research [2] which is influenced by the STEAM middleware, presents whats described as a lightweight middleware for mobile Java events named PENCA (Proximity-based Event-Notification for Collaborative Java Applications). Its design is partially influenced by the middleware STEAM as described in the previous section. It aims to provide a middleware that enables a Proximity-based Java Event system for Collaborative Mobile Applications. It is also designed to extend the Distributed Event Service of the Jini protocol [28] by enabling it for mobile ad-hoc networks.

Its primary aim is to provide proximity-based services that allows service providers to define the area of relevance of their services. The idea is that this helps to reduce the system complexity by limiting the number of participating nodes to those which are within the defined proximity. PENCA uses group communication in order to disseminate events to a specific multicast group, where the particular group is dependent on the event type and the proximity. The result of this design is a reduction in both bandwidth and computational resource consumption. It also allows loosely-coupled communication by allowing nodes to participate without being identified.

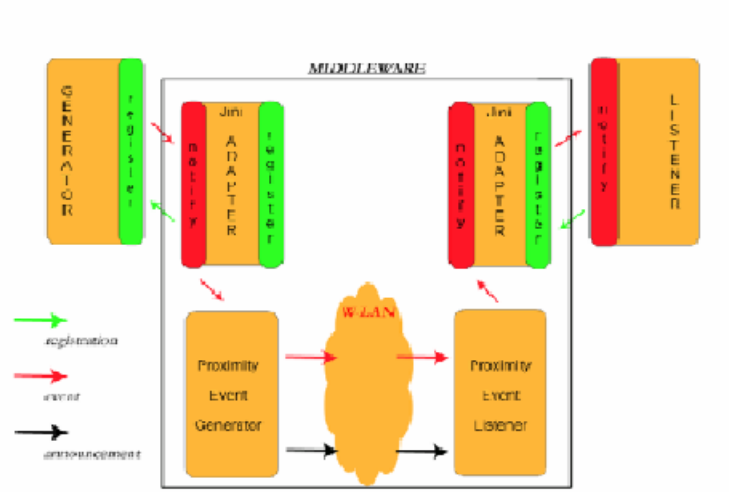

Figure 2.2: PENCA Architecture [2]

Figure 2.2 above shows the high level architecture of the PENCA middleware. The listener and generator components represent Jini Applications that utilise the PENCA Jini adaptors which provide an interface to the middleware for a Jini application. These adaptors have the effect of extending the Jini framework to support collaborative interaction in mobile ad-hoc networks.

While the design of PENCA is influenced by that of STEAM, it is focused on achieving a more lightweight solution. It removes some of the complexities of STEAM and is intended to be a middleware layer per application rather than per machine. The middleware is instantiated for each event type. This removes complexity in terms of having to manage multiple event types and has the effect that a single instance is only aware of a single (its own) proximity group. Multiple instances of PENCA are capable of working in a concurrent manner on a single machine. PENCA requires only the subject or event

type in order to publish. Also, no knowledge of the data structure is required. This allows an application developer complete freedom in terms of the data structure of event messages. STEAM is aware of the events data structure and so is capable of performing sophisticated filtering, whereas PENCA places the onus on the application developer to develop custom filters, specific to their particular requirements.

The PENCA middleware architecture consists of five main components that provide the core functionality and services of the middleware. The Location component's function is to provide an interface for a location service. A location service uses sensor data in order to provide real-time geographic location information to the application. Proximity is represented as a geometric shape using a defined shape, dimension and reference point. The design allows for any geometric shape to be used to define a proximity. This component also provides functionality to determine whether a given location is within a defined proximity. This is used by the middleware when making decisions regarding message delivery.

The Group Communication component is responsible for providing proximity based adhoc communication and also event delivery. It assigns proximity groups based on its internal hash algorithm. Announcement and event messages are also disseminated by this component. On the consumer side, this component is responsible for joining and leaving proximity groups. It also processes received messages and forwards them if necessary.Multicast messaging is used to disseminate events. Similarly to STEAM, this enables a high level of scalability for systems using the middleware.

The Event Handler and Wrapper components provide interfaces to enable Java and Jini applications respectively to utilise the middleware. The wrapper component provides a wrapper around the event handler component and acts as a Jini adaptor in order to enable a Jini application to interface with the middleware.

The Filtering component is a lightweight component that provides an interface to enable the provision of custom filters which can then be applied by the middleware. As discussed earlier, the design of PENCA is to allow the application developer to develop custom filters. This component allows any such filters to be applied within the middleware once they implement the required interface.

Chapter 3

Design

This chapter outlines the architecture and design of the context-aware advertising system. Initially high level architecture and design goals for the system are presented. This is followed by a requirements analysis, a more detailed system architecture design and a discussion on key design decisions and considerations.

3.1 Design Goals

The high level goal for this project is to deliver a context-aware advertising system, deployable in wireless ad-hoc networks. This system should allow multiple consumers and advertisers to come together at a location to form a collaborative application. Network connections should be formed in a dynamic ad-hoc manner. Advertiser and consumer nodes should also be free to enter and exit the application space without affecting the overall operation of the application. The collaborative nature of the application requires that consumer and advertiser nodes are not required to have prior knowledge of other application nodes. It is envisaged that the system could potentially be used by users with an IEEE 802.11 enabled mobile computing device with built-in GPS (Global Positioning System) capabilities, such as a PDA (Personal Digital Assistant).

Figure 3.1 illustrates the high-level concept behind the context-ware advertising application. Two advertisers and three consumers have come together in a spontaneous

manner and formed an ad-hoc network. The two advertisers send advertisements which are disseminated across the network to the participating nodes, using multi-hop messages to reach all consumer nodes. The coloured arrows indicate the route taken by the advertisements in order to reach the different consumers. There is potential for the application space to cover a wide geographic area depending on the number of application nodes that are present.

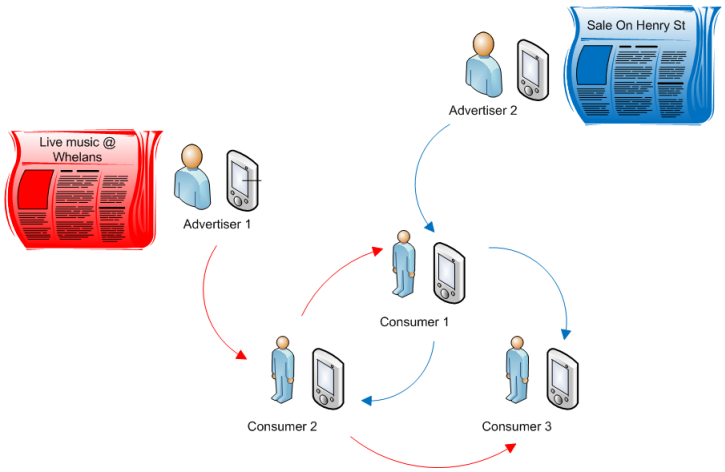

Figure 3.1: High Level System Concept

As part of the initial proposal, the system is to utilise and build on existing work, specifically a lightweight Java middleware component named PENCA, as discussed in Chapter 2. The PENCA middleware provides a proximity-based Java event framework for mobile applications. It provides an API (Application Programming Interface) that can be utilised by both producer and consumer applications. As PENCA is implemented in Java, this project is also to be implemented in Java in order to utilise this middleware. It is intended to build an additional domain-specific advertising layer that interfaces with the Jini adaptors provided by PENCA. This additional layer should provide additional services to enable context-aware advertiser and consumer applications to be built. The scope of this project also includes the design and implementation of consumer and advertiser applications that utilise this advertising layer, as a means of demonstrating and evaluating the additional layer. The advertising layer has been named CAAF (Context-Aware Advertising Framework), which is referred to throughout the remainder of this document.

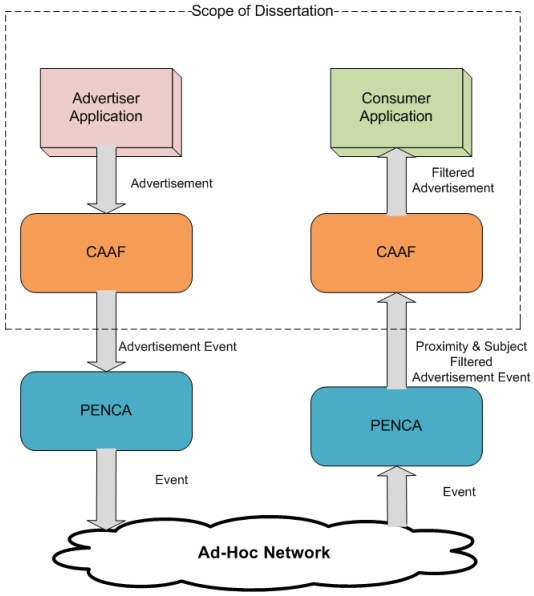

Figure 3.2: High Level Architecture

Figure 3.2 above illustrates at a high level, the overall architecture for the context-aware advertising system and the components which are within the scope of this project. There are three essential components to be designed and implemented as part of this project:

- CAAF Advertising Layer: This layer builds on the services provided by PENCA and provides further services specific to the advertising domain. It interfaces with the Jini adaptors of the PENCA middleware in order to utilise the distributed event services provided by it. It also provides an API to allow advertiser and consumer applications to utilise the advertisement services provided by this layer.
- Advertiser Application: This component provides an end user application and GUI (Graphical User Interface) for advertisers to create, publish and manage advertisements. The advertiser application interfaces with the CAAF layer in order to use the services provided by this layer.
- Consumer Application: This component provides an end user GUI application for consumers. It enables them to subscribe to the advertising service and express an interest in receiving advertisements. It displays advertisements to consumers and also allows them to manage their preferences and received advertisements.

Following the state of the art review, a number of key characteristics and features were identified as being crucial to the design of the system. Central to the design was the idea that the system should aim to provide a high degree of personalisation and deliver only targeted and relevant advertisements to consumers. The early high-level design goals and features identified were as follows:

- Flexible and Extensible design. The CAAF middleware is domain-specific in that is intended to be used by advertising applications. However, it should aim to be flexible and provide a degree of freedom and extensibility to advertising application developers wishing to utilise it.
- The Advertiser application should enable advertisers to target their advertisements at specific consumer demographics. It should also be flexible and be usable in multiple application scenarios.
- The Consumer application should achieve high degree of personalisation and aim to only deliver highly targeted advertisements to the user. The application should be un-intrusive and should also be designed with user privacy concerns in mind.
- The CAAF layer should provide an easy to use API for application developers, providing an abstraction that hides its underlying details.

3.2 Requirements

Initially, a use case analysis was performed in an attempt to determine how advertisers and consumers would potentially interact with the system. This was used to create a set of requirements to determine the functionality that should be provide by both applications and also the CAAF layer in order to meet these requirements.

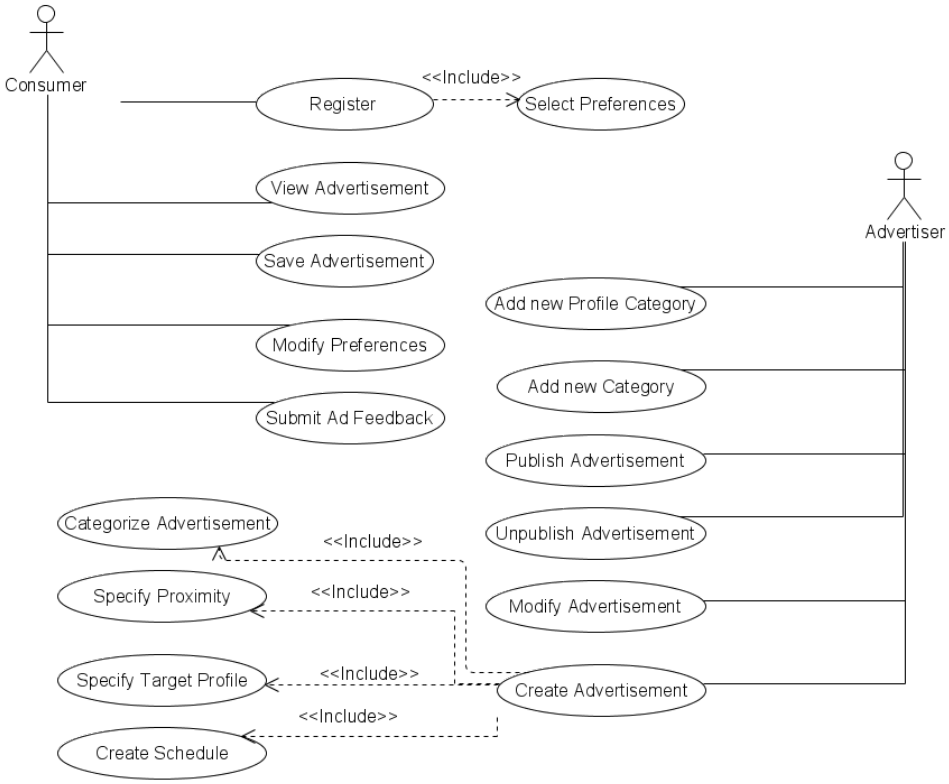

Figure 3.3: System Use Cases

Figure 3.3 shows the use cases that were developed as part of the system design process. The use cases identify the functionality that should be provided by the consumer and advertiser applications and were subsequently used in the design of the CAAF middleware layer to help determine the services and interfaces that should be provided.

The consumer application's core functionality is to allow the user to register for the service to express an interest in receiving advertisements and to view an advertisement presented to the user. The registration process involves selecting preferences in order to increase the level of personalisation and to improve the relevance of the advertisements delivered to the user. A consumer also has the option of saving or submitting feedback on received advertisements.

The core functionality of the advertiser application is the creation and publishing of advertisements. As part of the ad creation process, the advertiser can specify a number of associated attributes including category, proximity and target user profile. The prox-

imity allows them to specify the geographic range within which the ad will be delivered while the category and profiling options allow them to target an advertisement to a particular consumer type. The advertiser is also able to create new categories and profile data if necessary, in order to meet their particular requirements. A schedule can be created in order to specify when the advertisement should be published e.g. at a particular time of day, particular date etc.

3.3 Design Features

This section discusses the main design features that have been developed in order to meet the outlined requirements.

3.3.1 Increasing Advertisement Relevance and Targeting

Following the state of the art review of similar advertising systems, it is apparent that a major common theme across all of them is an attempt to maximise the relevance and targeting of any delivered advertisements. As discussed previously, this is particularly important with mobile advertising due to the personal and intimate nature of mobile computing devices. One of the key design considerations for this system therefore surrounds how to achieve this desired degree of personalisation and relevance.

3.3.2 Proximity

The PENCA middleware provides proximity filtered events in a mobile environment. The first step therefore is to determine how best to utilise these services. An instantiation of the *PENCAJap* (publisher) class requires a proximity to be defined. Therefore, a proximity can be defined for every event type, which in this case is an advertisement. The system design enables an advertiser to define a proximity for each advertisement, giving them control over the geographic range over which to transmit each advertisement. The *PENCAJac* (consumer) class does not require a proximity input. Proximity calculations are based only upon the advertiser's defined proximity. It was decided to add functionality to enable a consumer to also define their desired proximity range.

While an advertiser may potentially define a large proximity range to deliver advertisements within, the consumer may only be interested in receiving advertisements within a much smaller proximity range. This feature gives them the ability to define their own personal range. Figure 3.4 illustrates a potential scenario where the consumer is within the advertiser's defined proximity, but the advertiser is outside the consumer's proximity. An advertisement event received in this scenario would not be delivered to the consumer.

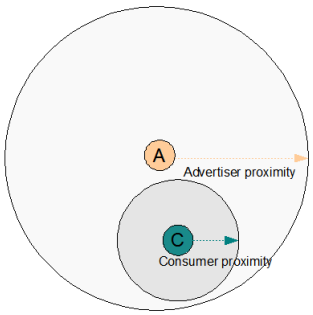

Figure 3.4: Advertiser and Consumer Proximities

3.3.3 Frequency

Another important feature of the design is the notion of frequency. On the advertiser side, this refers to the frequency of dissemination of advertisements. The PENCA middleware in fact sends two types of messages per event: event announcements and the events themselves. The event announcements notify potential consumers of event types that are being published. A consumer decides to join a proximity group and subsequently receive events of this type, based on the contents of this announcement. The event message contains the content of the event, in this case an advertisement of some kind. There are two parameters used to set the frequency of both these event types: event interval and announcement interval. They simply define the interval (in milliseconds) between the sending of each event. The design allows for these values to be configured by the advertiser application developer.

Another important design feature for the consumer side is the ability for a user to define the frequency at which they receive advertisement alerts. For example, if a number

of advertisers were disseminating advertisements within the same geographic space, a consumer may potentially be inundated with an unacceptable number of advertisements. This feature allows the consumer to limit the frequency at which they are notified of advertisements to what they deem to be an acceptable level.

3.3.4 Scheduling

Typically an advertiser will want to restrict the publishing of an advertisement to a particular time frame (e.g. only at a particular time of day or day of week). For example, a shop using the service may only want to send out advertisements during business hours. The system design includes a feature enabling the advertiser to specify this type of publishing schedule per advertisement. This in effect enables an advertiser to create an advertising campaign schedule including an exact time frame to publish each advertisement.

3.3.5 Event Types and Categorisation

As detailed previously, PENCA enables proximity and subject based filtering in events. In order to send events using the PENCA layer, a producer must select an appropriate event type. The event type is a String data type and is intended to act as an identifier or clue to the type of content included in the event. Similarly, in order to subscribe to receive or be notified about events, the consumer must specify a particular event type to subscribe to.

A design decision was required regarding how to define and choose event types for advertisements. One option would be to use one pre-defined event type for all advertisements. This would allow advertisers and consumers to publish and subscribe all advertisements according to this pre-defined event type. However, this would result in a consumer receiving all advertisements within an application space, thus requiring extensive consumer-side filtering for every received event. In order to reduce the level of processing and filtering required by a consumer, a decision was made to use the event type parameter as a more fine-grained means of categorising an advertisement. Every advertisement has an associated category which is selected as part of the advertisement

creation process. Each advertisement's category directly maps to its event type when publishing using the PENCA layer. On the consumer side, the same notion of categories is used for event subscription. A consumer selects a category(s) in which it is interested in receiving advertisements. The consumer will then be subscribed as a listener to remote events of this type and will therefore only receive advertisements in this particular category.

Two key design issues arise from this decision to use the event type as a means of categorisation or filtering:

1. How to structure the category list in order to provide meaningful and useful categorisation.
2. How do consumers and advertisers discover and become aware of new categories and event types.

A decision was taken to use a hierarchical category tree structure. A category tree structure was in preference to other commonly used categorisation techniques (e.g. tagging). The primary motivation for this is to enable a direct mapping to the event type. A particular category's event type is determined by tracing the path to its root node. Similarly, when a consumer subscribes to receive this type of advertisement, their category selection will be mapped to create a subscription to that particular event type. The structured nature of a category tree enables this type of one-to-one mapping which may not be possible with other categorisation techniques.

The category tree starts with the root category, and all other categories stem from this root category. The category tree provides a flexible structure in that it allows any number of categories and sub-categories to be created. It also provides a useful means of managing and visualising a large amount of categories.

When creating an advertisement, an advertiser selects an appropriate category from the available category tree. As discussed, this category selection is subsequently mapped to become the PENCA event type for this particular advertisement. The event type value is determined by tracing the path from the category tree root to the leaf i.e. the selected category. The event type is composed of a comma delimited string, where each value represents a node along the path to the selected category.

On the consumer side, the same category tree structure is provided to consumers. They can select categories which are of interest to them and which they are interested in receiving advertisements in. A consumer can select multiple categories and can also select category nodes at higher levels in the tree hierarchy. i.e. they can select 'parent' categories that have a number of sub-categories. Thus by selecting these parent categories, a consumer can express an interest in receiving advertisements in all its sub-categories.

Figure 3.5: Category Tree

Figure 3.5 provides an example of a simple category tree. The root node 'Categories' contains three children or sub-categories. The sports sub-category has two further sub-categories. Using this type of hierarchical tree structure, it is possible to extend the tree to any number of levels. In an example scenario, a consumer wishing to receive football advertisements would need to express an interest in this category. They could explicitly select the football category or alternatively could subscribe to a parent category, which in this case would be sports. A subscription to the sports category would allow the consumer to receive advertisements related to any of its sub-categories. This feature could be particularly useful in a larger category tree containing multiple levels of categories.

While the category tree provides a useful mechanism for both advertisers and consumers to publish and subscribe to particular types of categories of advertisements, it also raises some issues regarding the discovery of any newly created categories. It would be neither possible nor practical to attempt to create a global category tree that provides all potential categories for advertisements in all domains. Instead, the idea is to give the advertisers the ability to add new categories to meet their particular domain requirements. In order for consumers to become aware of newly created categories, a notification or discovery service must be included in the design. To achieve this, the system design includes a default category event which is disseminated at regular intervals,

and includes category information. Essentially it is an advertisement event of available categories. The event type is well known and advertisers and consumers are configured with knowledge of the event type. By default, consumers subscribe to this well known event and advertisers publish their current category tree using this category event type. When a consumer receives a category event, it scans the contents and searches for any new categories. For any new categories, the consumer will be automatically subscribed if they are already subscribed to its parent category. Otherwise, the category is just added to the consumer's category tree and made available to the consumer. In order to improve efficiency and minimise the size of data transmitted, advertisers send a sub-tree of the overall category tree. The sub-tree contains only new categories that the advertisers have themselves added. Each advertiser maintains this sub-tree in parallel to the overall category tree.

3.3.6 Custom Filters

In order to increase the filtering capabilities of the system, an additional custom filtering mechanism has been designed. Similarly to the design of the category tree structure, the design of this feature focuses on providing advertisers with the ability to dynamically create filters suitable for their particular domain. The design of the filtering component includes an interface that can be implemented to create custom filters. Three basic filters have also been designed as part of the middleware in order to test this design feature.

Any of the available filters can then be included with an advertisement to be applied at the consumer side. For example, an advertiser could create an 'over 18' filter to target its advertisement at this demographic. A consumer would then have to explicitly select this 'over 18' filter in order to receive that particular advertisement.

Similarly to newly created categories, consumers must be capable of discovering any newly created filters in order to be able to select or subscribe to them. An advertiser's available filters are therefore included with the category event to allow consumers to discover any such filters. When a consumer received a category event, they can then process and update their category and filter lists if any new entries are present.

3.3.7 Category Event Versions

A versioning system has been designed to improve the efficiency with which the consumer can process and deal with category events. Each category event contains version information for the contained category tree and filter list. When a change is made to either, an advertiser increments the version number. The transmitted version information contains the advertiser's id and the latest version number for the tree and filter list. Upon receiving a category event, a consumer can then examine the version information to deduce whether the event contains any new information. The consumer can ignore received category events if it has already received that particular version, thus avoiding unnecessary processing. The advertiser's id is included with the version information in order to distinguish between category events from different sources. The consumer must therefore maintain a list of the most recent category tree versions it has received and processed.

3.3.8 Consumer Feedback

A feedback mechanism has also been included in the design, as a further means of filtering advertisements. The focus of the feedback feature is to provide a method of filtering the sources of advertisements. A consumer provides positive or negative feedback to the system regarding advertisements that have been viewed. Feedback provided is assigned to the source of that particular advertisement. The system can subsequently use this provided feedback as a means of filtering future advertisements from the same advertiser source. This feedback mechanism aims to provide one mechanism or counter measure against potential spammers or abusers of the system.

3.3.9 Data Mapping

Providing a means of storing data was a further requirement and design decision for the system. This is a requirement on both the advertiser and consumer sides. Advertisements, category trees, filters and user preferences are all required to be saved in some form in order to provide all of the systems functionality. XML (Extensible Markup

Language) has been chosen in order to achieve this. The main motivation behind this decision is the fact that it offers interoperability, readability, and platform independence. It is not as efficient as something like a relational database, particularly for storing and retrieving large amounts of data. However it is more lightweight in that it is basically just a text file. This is of particular importance in a mobile environment in which available resources are generally limited. XML Schemas are therefore required for all of the data that is required to be written to XML, namely advertisement lists; user preferences; category list and custom filters. By providing XML Schemas to represent this data, it provides another interface to the system for application developers. For example, a list of advertisements to be publish could be created in XML format. This XML data could then be processed at runtime resulting in the publishing of the contained advertisements.

3.4 Architecture Design

This section presents the architecture for the design of the CAAF layer and the consumer and advertiser applications. PENCA provides Java interfaces for applications to make use of the middleware. The design is based on using and extending this layer. The following sections present the design of each of the systems key components.

3.4.1 CAAF Layer

The CAAF middleware layer resides between the PENCA middleware and any consumer or advertiser application and is therefore central to the design of the overall system. Figure 3.6 illustrates the architectural design of the CAAF middleware. It illustrates the various architectural components and the interfaces and dependencies between these components. The Event component is responsible for interfacing with the PENCA layer. It implements the appropriate interfaces in order to act as a generator and listener to PENCA. The Location component is required in order to provide a location service which provides real-time geographic location information. The Data component provides a mapping between Java and XML data. The Scheduling component is used on the producer side and is responsible for scheduling and coordinating advertisement dissemination. The Filtering is used filter incoming advertisements

and makes decisions regarding delivery of advertisements. CAAF Manager component passes received events to the Filter component to determine how it should proceed with that event. The Filter component also processes the default category events and retrieves category and filter data from these events. CAAF Manager is the central or core component and is in effect the engine of the CAAF layer. It manages and co-ordinates interactions among the other components. The CAAF API component's role is to provide an interface that can be used by applications wishing to make use of the CAAF layer.

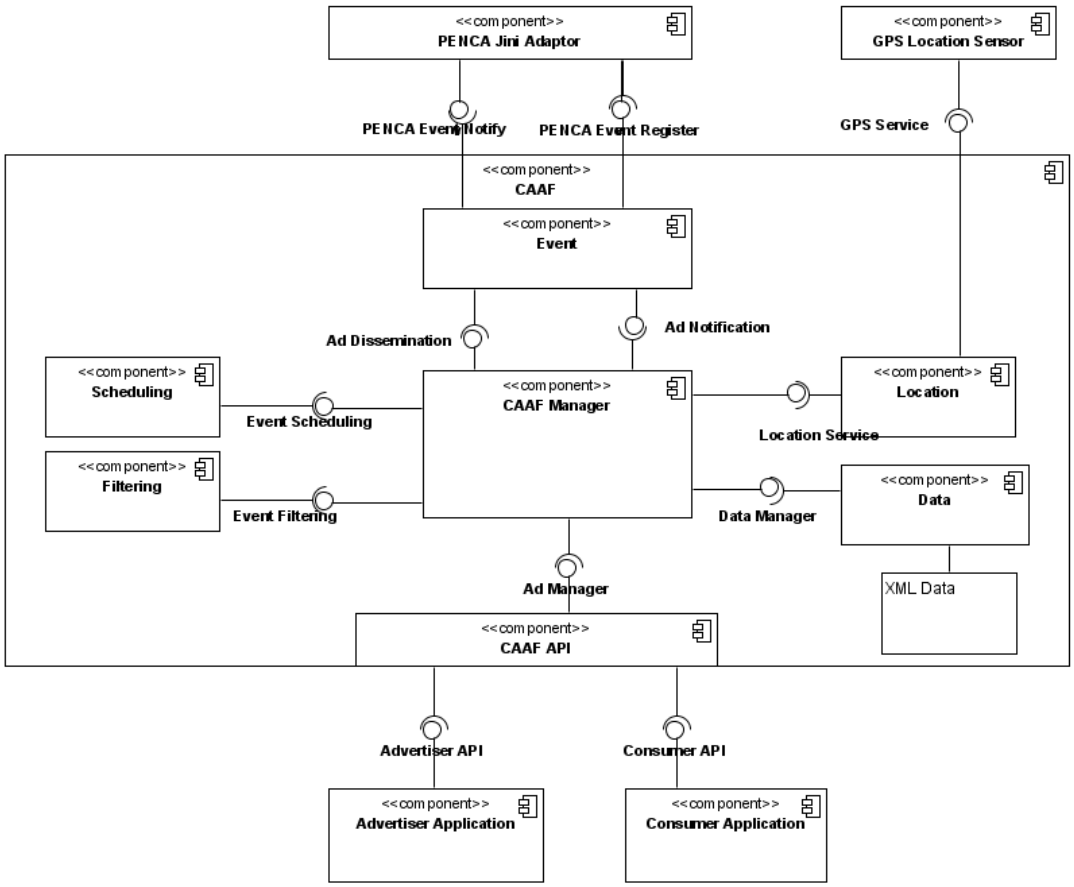

Figure 3.6: CAAF Layer Components

Advertisement Data Representation

Representation of an advertisement is a key design decision within the overall system design. An advertisement is at the core of all interactions in the system. From its initial creation and publishing by an advertiser, its dissemination across the network, and subsequent reception and viewing by a consumer. PENCA provides much flexibility in terms of event dissemination. Any class that implements the *java.io.Serializable* interface can be used as a PENCA event.

An abstract class and an interface have been designed to define the structure of advertisement data. There is a separation between the data that is required by the CAAF middleware in terms of the producer side, and the data that is required on the consumer side. In order to improve efficiency, only data that is necessary on the consumer side should be transmitted. The abstract class *CAAFEvent* represents the core data that is required by the CAAF middleware in order to transmit an advertisement event. A class that extends this class should be passed to the PENCA layer for transmission across the network. A combination of the*id* and *advertiserId* attributes are used to uniquely identify each advertisement. The advertiser id enables the identification of the source of the advertisement which can be used for filtering purposes. In addition, title, location, and time sent are required by the CAAF layer. The *CAAFEvent* also contains an attribute *filters* which is of the type *Collection<IAdFilter>*. It includes any filters that should be applied to that particular advertisement.

Figure 3.7 illustrates the structure of the *CAAFEvent* class. Two sub-classes have been designed within the scope of this project. The *CategoryEvent* is used internally within the CAAF layer to transmit category and filter information. The category tree is represented using Java's abstract *TreeNode* class. This provides a flexible way of representing the category tree as in a Java class. The attribute *filterList* in *CategoryEvent* contains a list of all available filters. The *AdvertisementEvent* class was designed for use by the consumer and advertiser applications that were also designed within the scope of this project. It provides attributes representing an advertisement's content and category in addition to what is provided by the *CAAFEvent* class. An application developer is free to extend this to meet their requirements.

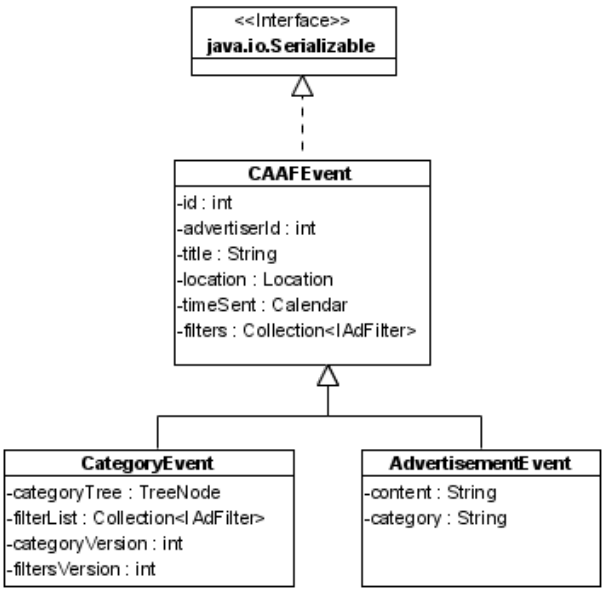

Figure 3.7: CAAFEvent

In order to utilise the CAAF layer, an application developer also needs to implement the *ICAAFAdvertisement* interface. The interface contains four methods which are required in order to provide all the required functionality on the producer side. The *getPublishingSchedule* method requires the return type to be a class implementing the *ISchedule* interface. This is required by the Schedule component. Every advertisement must be have an associated category in order for it to be disseminated as a specific event type. The is also the event type that the consumer side registers for. The *getCategory* method provides this information. The *getProximity* method is necessary in order to define a proximity range for each advertisement. The *getCAAFEvent* returns a class that extends the abstract class *CAAFEvent*. This is the class that will be disseminated to consumers. In the examples above, this refers to *AdvertisementEvent* or *CategoryEvent*.

Figure 3.8 shows the *ICAAFAdvertisement* interface and two classes that implement it. The *CategoryAdvertisement* is used to disseminate a *CategoryEvent* while the *Advertisement* class is used to disseminate an *AdvertisementEvent*. An advertiser application using the CAAF API must pass a class implementing *ICAAFAdvertisement* in order to create or publish an advertisement.

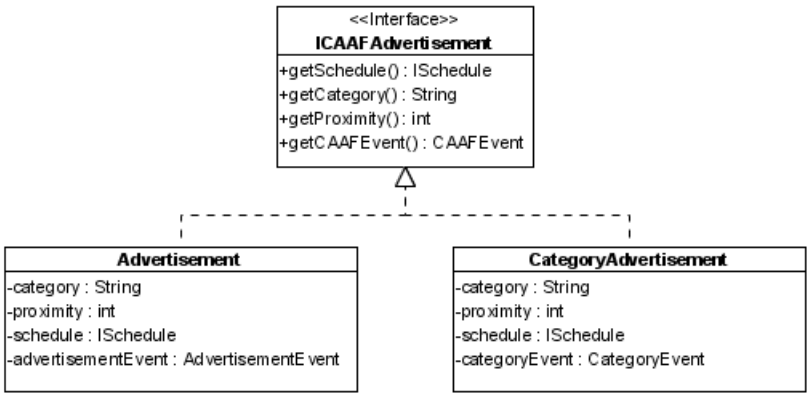

Figure 3.8: ICAAFAdvertisement Interface

Event Dissemination and Handling

The CAAF Manager component is the central component in the CAAF layer. It coordinates and interacts with the other components and is central to any operation in the system. Requests to publish advertisements are handled by this component, as are incoming advertisement notifications. The CAAF Manager component also interacts with the Data component in order to retrieve or update persisted data. When an advertiser publishes an advertisement, it creates an *EventScheduler* instance in order to achieve this. Similarly, it creates instances of the *EventHandler* class to enable a consumer to register as a listener for a particular category of advertisement. It also retrieves a Location Service which is required in order to instantiate PENCA's Jini adaptors.

The Event component interfaces directly with the PENCA middleware, specifically the Jini adaptors provided by the wrapper component of PENCA. This component is therefore responsible for instantiating PENCA and must also implement the *RemoteEventListener* and *IEventListener* interfaces. In order to register producers and listeners with PENCA, it needs to call the *register* and *notify* methods provided by the adaptors. PENCA is designed to be lightweight and a single instance is created per event type. The *EventHandler* and *EventDispatcher* classes are directly responsible for these instantiations. An *EventHandler* instance is created for each category that a consumer subscribes to, which subsequently creates a corresponding PENCA instance for this

event type. The *EventManager* manages a list of *EventHandler*s that have been created to listen for different advertisement categories. For every notification received, the event manager is notified and it processes the event accordingly.

The*EventScheduler* and *EventDispatcher* classes are concerned with scheduling and dissemination of advertisement events. The *EventDispatcher* class uses the *notify* method of the PENCA Jini adaptor to initiate the dissemination of an advertisement. The *EventDispatcher* class extends the abstract *SchedulerTask* class. This enables advertisement event dispatching to occur according to the defined publishing schedule.

The Schedule component provides a flexible framework to enable advertisements to be published according to a detailed schedule, and is based on a framework outlined in [29]. It builds on Java's in *Timer* and *TimerTask* classes which provide basic scheduling capabilities. Event dispatching is the task that is to be scheduled. Therefore, the *EventDispatcher* extends the abstract *SchedulerTask* class to enable this. The class *PublishingSchedule* is used to define an advertisement's publishing schedule. The *ISchedule* interface defines the methods that a schedule passed to the CAAF layer should implement. It extends the *ScheduleIterator* interface which is required by the *Scheduler* class. The *EventScheduler* class creates an instance of the *Scheduler* class and calls its *schedule* method using the parameters of an *EventDispatcher* instance and its associated *ISchedule*. The *Scheduler* class then takes care of starting or running the event dispatching tasks as appropriate. The class diagram in Figure 3.9 illustrates the classes involved in advertisement event scheduling and the relationships between them.

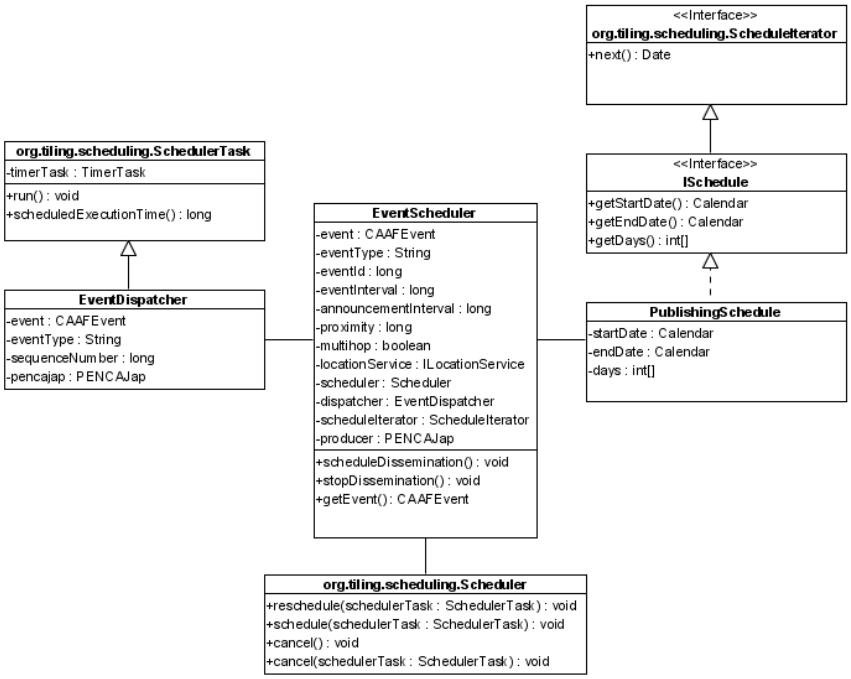

Figure 3.9: Advertisement Event Scheduling

Filter

The Filter component is concerned with providing additional filtering at the consumer side (i.e. additional to the proximity and category filtering). When the CAAF Manager is notified of a new advertisement, it passes the advertisement to this component. Based on its internal filtering rules, and the filters contained within the advertisement, it notifies the CAAF Manager whether or not the consumer should be notified of the advertisement. This component also deals with incoming category events.

When advertisement event is received, it goes through a number of filtering steps. If it fails any of these steps CAAF Manager is notified that the advertisement should not be delivered to the consumer. There are five filtering steps involved in this process:

1. Already Alerted: If the consumer has already been alerted of this particular advertisement, then it is filtered. It is assumed that a consumer will only wish to view

a particular advertisement once. Advertisements are uniquely identified based on a combination of the ad id and the advertiser's id.

2. Outside Proximity: The current proximity to the advertisement is calculated based on the consumer's current location and the location contained within the advertisement event. If the calculated value is greater than the consumer's proximity preference, then it is also filtered.

3. Frequency: Checks if the advertisement has been received within the consumer's frequency limit.

4. Negative Feedback: Advertisements originating from sources with negative feedback are filtered and not delivered to the consumer.

5. Custom Filters: Any filters contained in the advertisement are examined and compared with the consumer's selected filters. The consumer's preferences must contain a matching filter for each of the included filters within an advertisement. Otherwise, the advertisement is filtered.

Figure 3.10 shows the *IAdFilter* interface that is required to be implemented by custom filters. Three custom filters have been created within the scope of this project, which are also shown. The advertiser application allows filters to be created using any of these types. Further filters could also be developed to provide more advanced filtering.

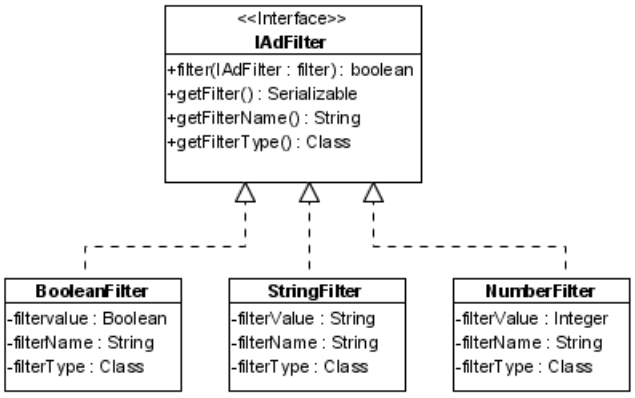

Figure 3.10: Filter Interface

The Filter component is also responsible for processing default category events received by the CAAF Manager. The following steps are taken when a category event is received:

- Check the version number of both the category tree and filters list.
- The version numbers are compared to a maintained list of version numbers that have been received. If the contained versions have already been received, the event is not processed any further.
- If the category version has not been previously received, the category tree is scanned. Every node is compared to the nodes in the existing tree. If a new node is detected, it is added to the existing tree. When complete, the version number is updated in the consumer's maintained list.
- If the filter version has not previously been received, it is scanned and compared with the consumer's existing filter list. Any new filters are added to the list and the version number is subsequently updated.

Location

The Location component is a lightweight component in the system. Attaining location information is a basic requirement for the operation of the overall service. In order to instantiate the PENCA middleware an instance of a class implementing the *ILocation-Service* interface is required. This is intended to be implemented by a class that obtains geographic data from a GPS Sensor but can in reality be provided by any class implementing this interface. The Location component of the CAAF layer provides an interface for other components to retrieve an instance of a location service. This is required on both producer and consumer sides to perform proximity calculations.

CAAF API

The CAAF API component provides an interface or a set of methods that can be used by consumer and advertiser applications. Figure 3.11 illustrates the methods provided by the *CAAFAPI* interface. Consumer and advertiser applications can utilise the services provided by the CAAF layer by using these methods. The method *addObserver*

is provided to allow a consumer application to register as an observer of the incoming *AdQueue*. When a consumer application is instantiated, an *AdQueue* is created. Any incoming advertisements that are not filtered, are added to this queue. Any registered observers are then notified or alerted of new advertisements. The application can then notify the end user or deal with the new advertisement alert according to its own design.

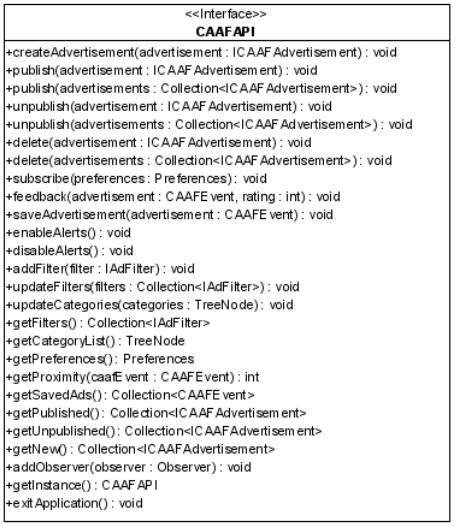

Figure 3.11: CAAF API Interface

Data Manager

The Data Manager component is responsible for storing and retrieving saved data. This is necessary to enable data such as published advertisements, user preferences etc. to be saved and retrieved at runtime. As discussed earlier, data is stored in XML format. It is the responsibility of this component to map the Java class information to XML and vice versa. This mapping must conform to the defined XML schemas. An interface is provided to enable this data to be retrieved by other components. The CAAF Manager component uses this interface in order to retrieve its required data when it is initialised. Similarly, the CAAF Manager component updates all necessary data and issues write requests to the Data Manager component when it is being closed or terminated. The interface provided by the Data Manager is illustrated in Figure 3.12.

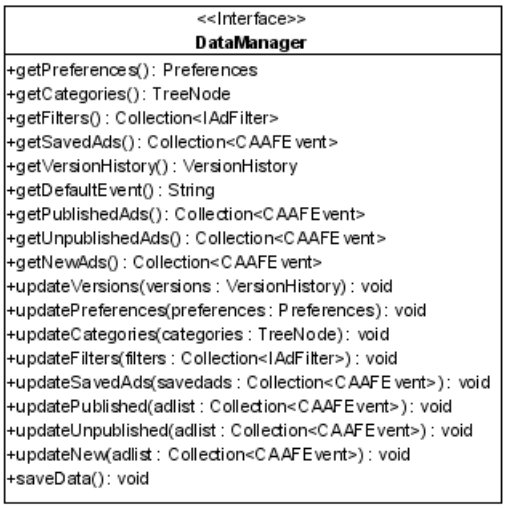

Figure 3.12: Data Manager Interface

3.4.2 Advertiser Application

The scope of the project also includes the design and implementation of an advertiser application. The intention is to create an application that could potentially be used by an advertiser to publish advertisements. The aim is to demonstrate how the CAAF layer could be used by an application developer. The application is also to be used as part of the evaluation of the project.

The advertiser application provides a GUI that end-user advertisers can use to create and manage advertisements. It also provides an interface to manage categories and filters. The main functionality provided to advertiser end user is:

- Advertisement Creation, Modification and deletion
- Advertisement Publishing and Unpublishing
- Category and Filter creation

The advertiser application uses the CAAF API in order to access and make use of the services provided by the CAAF layer (i.e. ad creation, ad publishing, filter and category

creation). The advertiser application must pass an implementation of the *ICAAFAdvertisement* interface to the API, which in this case is the *Advertisement* class as outlined previously in section 3.4.1. This class contains the advertisement event object as well as its associated publishing schedule, proximity and category. The CAAF layer can then schedule and publish advertisements based on the contained attributes. Figure 3.13 illustrates the structure of the advertiser application. The main application component obtains an instantiation of the CAAF API and loads any configuration files. It also instantiates the main GUI for managing advertisements. This GUI provides an interface to enable users to: create a new advertisement; publish, unpublish or delete an advertisement; create a new category; create a new filter. The Ad Creation Wizard component provides an interface to simplify the creation of a new advertisement.

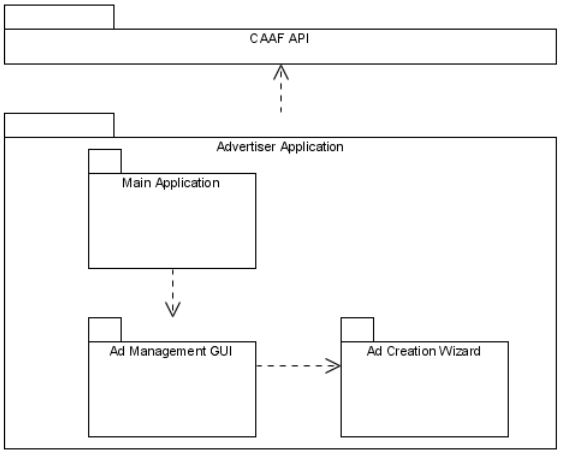

Figure 3.13: Advertiser Application

The creation of an advertisement involves a number of steps (ad content, categorisation, publishing schedule, filters). The wizard provides a means of separating each of these distinct steps. The Swing Wizard Framework [30] has been used in the design of the wizard component. It provides a flexible framework to assist in the creation of a Java Swing wizard. There are three classes from this framework that are used. Firstly, each page that is to appear in the wizard steps extends the abstract class *WizardPage*. There are five pages that compose the wizard steps. Each of these pages must therefore extend the *WizardPage* class and also implement the *getDescription* and *validateContents*

methods. Any user input is validated at each individual step. Each wizard page places user input into key-value pairs in a *java.util.Map*. The class *AdWizardResultProducer* is called when the finish button is selected. This class is required to implement the *WizardResultProducer* interface. It evaluates the contents of the *Map* and creates an *Advertisement* class to represent this data.

3.4.3 Consumer Application

The consumer application is intended to be used by users interested in receiving targeted advertisements to their mobile device. As discussed previously, the system aims to increase the relevance and personalisation of advertisements delivered in order to improve the overall user experience. There are essentially three main GUIs components that have been designed:

1. *ManagerGUI*: this is the central GUI component in the consumer application. It provides an interface for the consumer to manage the application. It enables them to view received advertisements, manage their preferences and also enable or disable advertisement alerts.

2. *PreferencesPanel*: this component is used by the *ManagerGUI* to enable users to select and modify their preferences. Categories of interest, frequency, proximity and custom filter preferences can all be chosen through this interface.

3. *AdContentPanel*: this component is responsible for displaying any received advertisements to the user. It provides a graphical representation of the information contained in an advertisement event.

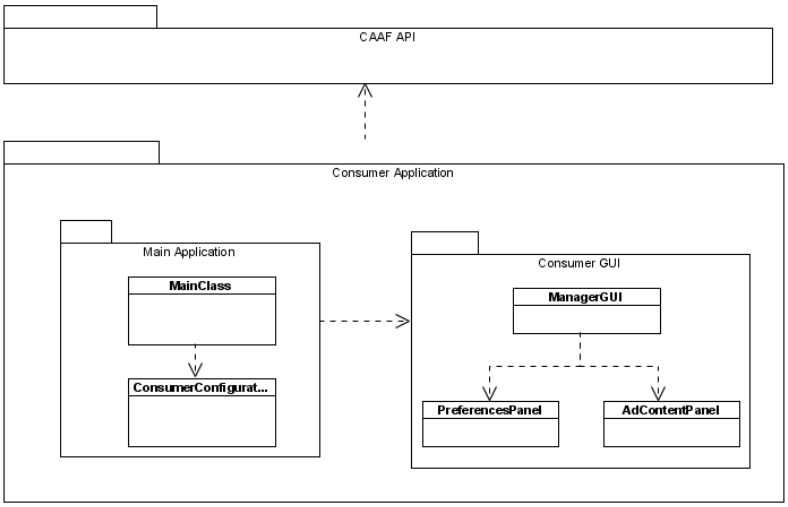

Figure 3.14: Consumer Application

Figure 3.14 illustrates the package structure of the consumer application. Similarly to the advertiser application, it interacts with the CAAF API in order to make use of the services provided by the CAAF layer. The *MainApplication* component manages the overall application. Its responsibilities include loading configuration parameters and also obtaining an instance of the CAAF layer. It also registers the consumer application as an observer or subscriber to the CAAF *AdQueue*. It is in effect following the observer or publish/subscribe pattern as described in [31], in which the consumer application is the observer and the event is the advertisement event. The consumer application is therefore automatically notified of newly received advertisement events.

Chapter 4

Implementation

This chapter describes the implementation phase of the dissertation. The implementation is intended to be a realisation of the design as outlined in Chapter 3. Initially the technologies used and development environment is discussed, followed by a discussion on the main issues in the implementation of each of the components. Some code listings are also included as a descriptive aid.

4.1 Development Environment

It is envisaged that the context-aware advertising system outlined would be deployed on highly mobile computing devices such as PDAs. For the purposes of the dissertation, a prototype application has been developed to demonstrate the system's potential in a mobile ad-hoc environment. The available hardware for testing purposes included two laptop computers equiped with wireless IEE 802.11 network cards, along with two Magellan GPS receivers. The dissertation aims included performing field experiments as part of an overall system evaluation. Therefore the prototype application has been developed to run on the available equipment and is not designed specifically for use on PDAs or other resource-constrained mobile devices. While Java does provide platform independence for its applications, applications developed for PDAs and other resource constrained devices typically require special focus on minimising the resource and memory usage of the application. Indeed there have been a number of developments such as

the Java Platform Micro Edition (Java ME), aimed specifically at providing an environment for applications running on mobile and embedded devices. The available resources in these types of devices are generally very limited. Java ME is basically a specification of a subset of the Java platform that is intended to run on these types of devices. Java Swing for example is quite resource intensive and would not therefore be ideally suited to building applications on these types of devices.

The main development environment used is the Java Standard Edition Development Kit (JDK) 6. There are also a number of additional dependencies of the project which are required as part of the development environment. The system uses the PENCA middleware so *penca.jar* is included to enable its interfaces to be used by the system. As discussed previously, PENCA extends the Jini event model and it therefore has a dependency on part of the Jini framework. Specifically the *start.jar*; *jsk-lib.jar*; and *jsk-platform.jar* Java archives are all required by PENCA and therefore also by the CAAF development environment.

The *GPSService* class is provided in PENCA to retrieve real-time GPS location information and requires the native Java interface for serial ports provided in *RXTXcomm.jar* archive. The native libraries *rxtxSerial.dll* and *rxtxParallel.dll*, which are specific to the Microsoft Windows platform are also required by the *GPSService*. The Log4j framework [32] is used for logging purposes across the system, *log4j-1.2.14.jar* is included as a result. Dom4j [33] has been used in order to support Java to XML mapping in the system, which requires dom4j-1.6.1.jar and jaxen-1.1.1.jar archives to be included.

The advertiser and consumer applications are developed using Java Swing to provide the GUI front end. This is included as part of the standard JDK. A number of additional libraries and frameworks have been used to assist in the development of different features of the GUI applications.

- *looks-2.1.4.jar* and *swing-layout-1.0.jar* are used to assist in the layout of components in the swing GUIs.
- *wizard.jar* provides a framework for building Swing wizards and is used by the advertiser application. [30]
- *JCalendar4.jar* is a small archive that is used to display a calendar date selection combo box. This is also used in the advertiser application. [34]

4.2 CAAF Layer

This section covers the key issues involved in the implementation of the CAAF advertising layer. Each of the components described in Chapter 3 are discussed.

4.2.1 Event

The Event component interfaces with the PENCA middleware and registers as an event publisher and listener. Two Jini adaptors are provided by PENCA which the event component instantiates. *PENCAJap* and *PENCAJac*. The *EventScheduler* class creates an instance of *PENCAJap* for each event type. A number of parameters are required for this instantiation, including an *ILocationService* instance and a *Shape* object to define the proximity within which events are disseminated.It also creates an *EventDispatcher* instance for each *PENCAJap* instance created.

```
producer = new PENCAJap(eventType,new Circle(proximity), filters,
    producerAtLeastOne,announcementInterval, eventId,
                                        multihop, locationService);
eventdispatcher = new EventDispatcher(eventObject,eventType, producer,
    sequenceNumber);
```

The *EventDispatcher* class extends the *SchedulerTask* class enabling its run method to be called by the Scheduling component when required. Its creates a *RemoteEvent* object in order to call the *PENCAJap notify* method.

```
MarshalledObject mo = new MarshalledObject(event);
RemoteEvent re = new RemoteEvent(event.getClass().getSimpleName(), pencajap.
    getJiniEventType(), sequenceNumber, mo);
pencajap.notify(re);
```

The *EventHandler* class creates an instance of the *PENCAJac* Jini adaptor for the consumer side and registers itself as listener of the event type used to create this instance. It is required to implement the *RemoteEventListener* interface in order register as a listener to remote events. The PENCA adaptor calls its *notify* method when a new remote event is received.The *EventHandler* class maintains a list of listeners to events of this type. Its *notify* method notifies any listeners of this new event.

```
pjac = new PENCAJac(eventId, listenerType, locationService, filters,
    listenerAtLeastOnce);                    pjac.register(this);
.......
public void notify(RemoteEvent arg0) throws UnknownEventException, RemoteException
    {
...
MarshalledObject marshalledObject = (MarshalledObject)arg0.getRegistrationObject();
Serializable object = (Serializable)marshalledObject.get();
for(IEventListener listener: eventListeners)
      listener.actionPerformed(object);
..
}
```

4.2.2 CAAF Manager

The CAAF Manager co-ordinates all interactions in this layer. It managers advertisement event publishing on the advertiser side and subscriptions on the consumer side. Upon initialisation, it retrieves saved data from the Data component. It also manages and initiates the dispatching of the regular category event that advertises available categories and filters. The *CAAFManager* class follows the singleton design pattern [31] to ensure that only one instance is created.

```
EventScheduler eventscheduler = new EventScheduler(caafevent, category,filters,
                                annoucementInterval,eventInterval, proximity,
                                    schedule);
eventscheduler.scheduleEventDissemination();
eventschedulerlist.add(eventscheduler);
```

CAAFManager implements the interface *IEventListener*. When a remote event is received by one of the event handlers, its *actionPerformed* method is called. The event is subsequently examined and dispatched to the Filtering component. If a positive result is returned by the Filtering component, the advertisement is added to the *AdvertisementQueue*, which extends *java.util.Observable*.

```
CAAFEvent ad = (CAAFEvent)p_event;
...
if(filtering.match(ad))
     adQueue.newAdReceived(p_event);

public class AdvertisementQueue extends Observable{
   ....
   public void newAdReceived(Serializable p_event, int eventNumber){
      adQueue.offer((AdvertisementEvent)p_event);
      setChanged();
      notifyObservers();
  }
```

4.2.3 Scheduling

The Scheduling component is used to schedule and run *SchedulerTasks*, which in this case are *EventDispatcher* instances. A *ScheduleIterator* is used to determine the next time that a task should be run, according to its defined schedule. The *next* method listed below contains the implementation that has been used to schedule advertisement event dispatching. It returns the next *Date* at which the task should be run, based on the contents of its schedule. The implemented schedule allows start and end times of day, start and end dates, and days of the week to be specified.

```
public Date next() {

/* Increment next scheduled time by the scheduled interval time until end hour
    reached, if the day is in the schedule
 */
if(nextTime.get(Calendar.HOUR_OF_DAY) < endTime.get(Calendar.HOUR_OF_DAY)
                &&(Arrays.binarySearch(days, nextTime.get(Calendar.DAY_OF_WEEK)) >
                    -1)){
        nextTime.add(Calendar.MILLISECOND, interval);
        return nextTime.getTime();
}

/* If end hour has been reached, set next scheduled start time & increment schedule
     times to next day in the schedule
 */
do{
        startTime.add(Calendar.DATE, 1);
        nextTime.setTime(startTime.getTime());
}while (Arrays.binarySearch(days, startTime.get(Calendar.DAY_OF_WEEK)) < 0);

/* Return next scheduled time or null to indicate schedule is complete
 */
if(startTime.after(endTime))
        return null;

else
        return nextTime.getTime();
}
```

The *EventScheduler* class is responsible for creating an *EventDispatcher*, *ScheduleIterator* and *Scheduler* instance. The *Scheduler* class calls the *schedule* method of *java.util.Timer* in order to run the task.

```
public class EventScheduler {
        private ScheduleIterator scheduleIterator;
        private EventDispatcher eventdispatcher;
      ......
        eventdispatcher = new EventDispatcher(eventObject,eventType, producer,
            sequenceNumber);
        scheduler.schedule(eventdispatcher, scheduleIterator);
```

4.2.4 Filtering

CAAFEvent objects that are received by the *CAAFManager* are passed to the Filtering component. The first step is to determine if the event is an advertisement or a category event. A separate class is used to process these two types of events.

Category event processing is performed by the *DefaultEventHandler*. It uses the version information contained in a category event as a means of eliminating unnecessary category events. A list is maintained of recent version numbers and their corresponding advertiser ids. The code excerpt that follows demonstrates the steps involved in processing a received category tree. *TreeNode* and its implementing class *DefaultMutableTreeNode* have been used to represent a tree structure in the code. A depth-first approach is used to compare the trees and discover any new categories.

Comparing filter lists is more straight-forward as they are a represented using a *Collection* of *IAdFilter*. As the category tree grows in size, the amount of processing required also increases. This is of particular concern with a mobile device which is likely to have limited processing power. The two attempts to minimise this processing are the versioning system already discussed and also the fact that advertisers send only a sub-tree of the full category tree. When a category or filter update is performed, the system updates its internal list of received updates to reflect this.

Advertisement events are passed to the *AdvertisementFilter* class. It returns a boolean value to indicate if the advertisement has passed all filtering successfully. Initially a number of core filters are applied, followed by any custom filters that are included with the received advertisement event. The core filters include filtering advertisements that: are outside the consumer's proximity range; have already been delivered; are from an advertiser with negative feedback; exceed the consumer's frequency limit. If the advertisement passes all of these filters, then any custom filters are checked. The code listing below shows how this is achieved.

```
private boolean customFiltered(){
        Collection<IAdFilter> adfilters = advertisement.getFilters();
        for(IAdFilter adfilter:adfilters)
        {
                // Get user filter of this type
                IAdFilter userFilter = getUserFilter(adfilter);
                if(userFilter == null)
```

```
private void scan(TreeNode existingtree, TreeNode newtree){
        int count = newtree.getChildCount();
        for(int i=0;i<count;i++)
        {
                TreeNode newnode = newtree.getChildAt(i);
                TreeNode existingnode = getExistingNode(existingtree,
                    newnode);

                if(existingnode==null)
                  {
                        existingtree.add(newchild);
                        autoSubscribe(newchild);
                  }
                else
                        scanForNewNodes(existingnode, newnode);
        }
 }
 private void scanForNewNodes(TreeNode existingtree, TreeNode newtree){
        Enumeration newchildren = newtree.children();
        while(newchildren.hasMoreElements())
        {
                TreeNode newchild = newchildren.nextElement();
                Enumeration existingchildren = existingtree.children();
                boolean alreadypresent = false;

                while(existingchildren.hasMoreElements())
                {
                        TreeNode existingchild = existingchildren.
                            nextElement();
                        if(nodeAlreadyPresent(newchild,existingchild))
                                alreadypresent = true;
                }
                if(!alreadypresent)
                {
                        existingtree.add(newchild);
                        autoSubscribe(newchild);
                }
        }
        scan(existingtree, newtree);
 }



                        return false;


                // If any of the filters do not match, return false
                if(!match(adfilter, userfilter))
                        return false;
        }
        // return true if all filters in the ad match
        return true;
 }
```

4.2.5 Location

Two location services have been used for this project. One class *GPSService* uses a Java program and driver written by Dr. Peter Barron of the Distributed Systems Group, TCD. The driver and program are compatible with the Magellan GPS 315, which has been used in experimental and evaluation set-ups for this research. The *GPSSimulator* written as part of the PENCA project [2] has also been used for testing purposes. The Location component therefore creates an instance of one of these classes according to configuration information which is retrieved from a configuration file. The *GPSSimulator* reads GPS co-ordinate data from a text file and uses this data as its location. An interval is also set to indicate how often it should update its location. The *LocationService* class provides a *getLocationService* method which returns the *locationService* object.

```
if(simulate)
        locationService = new GPSSimulator(simulationCoordinateFile,
            simulatorMethod, locationServiceInterval);
else
        locationService = GPSService.getInstance();
```

4.2.6 Data

This component provides a mapping from Java to XML data. It makes use of the Dom4j framework [33]. It is an open source framework and combines features of the Document Object Model (DOM) and Simple Application Programming Interface for XML (SAX). XML Schemas have been designed to store categories, filters, user preferences and advertisement lists. The listing below shows an example of data in a category xml file. Its structure is very simple and therefore very flexible. XML is also naturally suited to representing tree structured data. It can contain any number of categories which can themselves contain any number of sub-categories. The element name is mapped directly to the category name, meaning no additional element data is required.

Filters are represented using a single element which has attributes containing its type and value. The element text is the name or description of the filter. Three types of filters have been implemented but this can be extended.

```
<categories>
  <Sports>
    <football/>
  </Sports>
  <Fashion>
    <Ladies/>
  </Fashion>
  <Music>
    <Rock>
      <Indie/>
    </Rock>
  </Music>
</categories>
```

```
<root>
  <filter type="boolean" value="true">Over 18s</filter>
  <filter type="string" value="nds">fashion Store Gold Card code</filter>
</root>
```

The *consumerpreferences.xml* file contains a user's preferences and also a list of advertisements received and any feedback that has been submitted. A consumer can have subscriptions to multiple categories. These subscriptions are represented using a comma delimited string.

```
<preferences>
  <category>categories, Music, Rock</category>
    <category>categories, Fashion</category>
  </categories>
  <frequency>Always</frequency>
  <proximity>Town</proximity>
  <filters/>
  <alertsenabled>true</alertsenabled>
  <adsdelivered>
    <ad id="404" advertiser="1000"/>
    <ad id="406" advertiser="1000"/>
  </adsdelivered>
  <lastalert>Thu Sep 06 13:22:00 BST 2007</lastalert>
  <positivefeedback/>
  <negativefeedback>
    <ad id="406" advertiser="1000"/>
  </negativefeedback>
</preferences>
```

On the advertiser side, advertisement information is also stored in XML format. Advertisement content, category, proximity and scheduling information are also stored. Below is an example of the XML data for one advertisement. It provides a readable and self-descriptive format that could be used by an advertiser to create a list of advertisements that they wish to publish.

The Data component reads and writes persistent data to XML files in this format and

```
<advertisement>
  <id>405</id>
  <advertiserid>1000</advertiserid>
  <title>fashion store</title>
  <content>10 euro off with this advertisement</content>
  <category>categories, Fashion, Ladies</category>
  <proximity>10</proximity>
  <schedule>
    <startdate>Mon Sep 10 00:00:00 BST 2007</startdate>
    <enddate>Thu Oct 04 23:00:00 BST 2007</enddate>
    <monday>true</monday>
    <tuesday>true</tuesday>
    <wednesday>false</wednesday>
    <thursday>false</thursday>
    <friday>false</friday>
    <saturday>false</saturday>
    <sunday>false</sunday>
  </schedule>
  <filter type="string" value="nds">fashion Store Gold Card code</filter>
</advertisement>
```

maps them to Java objects for use by the CAAFManager class, hiding the details of this mapping from any other components.

4.2.7 CAAF API

This component provides an interface for applications to use the services provided by the CAAF layer. Its constructor gets an instance of the *CAAFManager* class. It is a relatively lightweight component and its main functionality is to call appropriate *CAAFManager* methods when one of its methods is called by an advertiser or consumer application.

4.3 End user Applications

Consumer and advertiser prototype applications have also been developed, both of which use the set of methods provided by the CAAF API component. As already discussed, the testing environment included laptop computers. The applications are therefore implemented with this operating environment in mind.

4.3.1 Advertiser Application

A Java Swing graphical interface is provided by the advertiser application. The *AdManagerPanel* provides the central interface that allows the user to create, manage and publish advertisements. A user can view lists of advertisements in the categories of published, unpublished and newly created advertisements. Figure 4.1 shows the GUI interface that is provided. Users can toggle between the various advertisement lists and publish or unpublish advertisements. An interface is also provided to view and edit an advertisement's data.

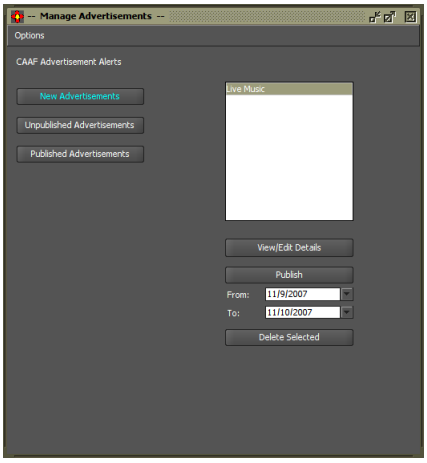

Figure 4.1: Ad Manager Panel

In order to facilitate the creation of an advertisement, a wizard is provided. The wizard was built using the Wizard Framework [30]. Figure 4.2 shows one of the wizard pages that a user completes during the advertisement creation process.

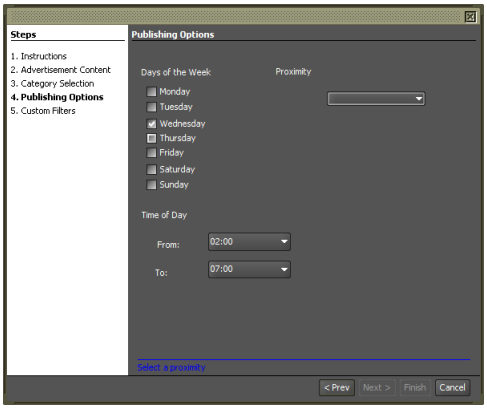

Figure 4.2: Ad Wizard

The *AdWizardResultProducer* class creates an advertisement object based on the user input once the finish button has been selected. User input is validated at each step of the wizard. The *Wizard* class is used to initialise the wizard and is responsible for returning an *Advertisement* object to its caller. The *initWizard* method of this class is listed below:

```
public Advertisement initWizard(){
              InformationPanel infoPanel = new InformationPanel();
              CategoryPanel categoryPanel = new CategoryPanel();
              PublishingOptionsPanel publishingPanel = new PublishingOptionsPanel
                  ();
              AdContentPanel contentPanel = new AdContentPanel();
              FiltersPanel filtersPanel = new FiltersPanel();

              Class[] clazz = new Class[] {infoPanel.getClass(), contentPanel.
                  getClass(),
                              categoryPanel.getClass(), publishingPanel.
                                  getClass(), filtersPanel.getClass() };
                                                      WizardResultProducer
                                  resultProducer = new AdWizardResultProducer
                                  ();
              Wizard wizard = WizardPage.createWizard(clazz, finishIt);
              Advertisement result = (Advertisement)WizardDisplayer.showWizard (
                  wizard, new Rectangle(600,500));
              return result;
}
```

Configuration information for the application is stored in an XML properties files. This information is loaded at runtime by the *AdvertiserConfiguration* class and can then be retrieved by the main application. Information such as event interval, type of location service, advertiser id, proximity mappings etc. are all set in these properties files. This provides greater flexibility in terms of changing or configuring the application and separates these settings from the compiled Java code.

4.3.2 Consumer Application

The consumer application is intended to be run by users wishing to receive advertisement alerts. It provides a Java Swing interface to enable users to interact with the application. The *PreferencePanel* and *AdDetailsPanel* are illustrated in Figure 4.3. Users select categories in which they are interested in receiving advertisements. Further filtering settings for frequency and proximity are also set through this interface. If the consumer is aware of any custom filters, these are also presented to the user on this panel. the *AdDetails* panel is used when an advertisement is being displayed to the user. It also enables a user to save and/or submit feedback for an advertisement.

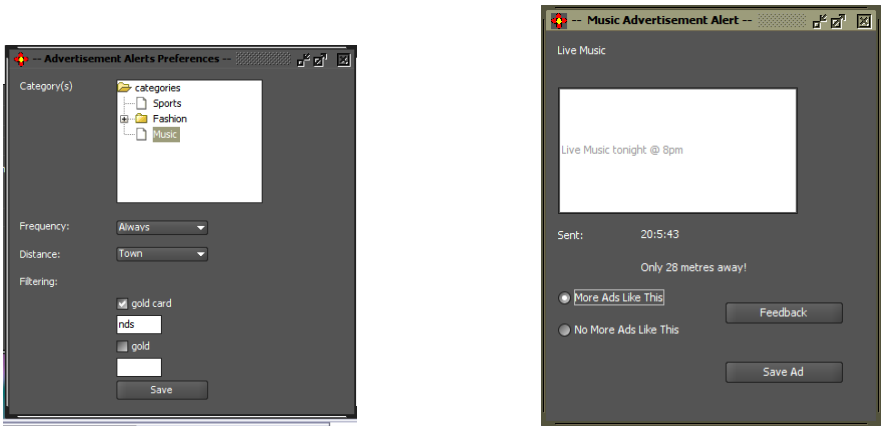

Figure 4.3: Preferences and AdDetails Panels

The *AdDetailsPanel* could be extended to provide other forms of content such as images. However resources such as available memory, bandwidth, display real-estate, and display capabilities tend to be limited in mobile devices. Therefore applications need to be designed specifically for its intended operating environment. Configuration information is stored in XML configuration files, similarly to the Advertising Application.

An important design goal of the consumer application is that it should attempt to be un-intrusive for the user and attempt to merge and integrate with the user's existing environment. The consumer application's main interface is implemented using the *java.awt.SystemTray* icon in an attempt to achieve this level of integration. The test environment for the application was on the Microsoft Windows XP Operating System. In this case, the *SystemTray* class is used to create an icon in the 'Taskbar Status Area'. However it also supports a number of other Operating Systems so it is not platform

dependent. When the consumer application is running, it creates an icon in the System-Tray. The user can manage the application from this icon by selecting from the various menu options. The preferences panel can also be loaded from this menu.

Figure 4.4: Main menu Interface and new Ad Notification

Figure 4.4 shows the main consumer interface and the available menu. It also shows the alert icon that is displayed when a new advertisement is delivered to the consumer. An information bubble displays a summary of the advertisement details including the title and proximity. If the user is interested, they can click to view the full advertisement. Received advertisements remain available from the main menu also, enabling a consumer to view them at a time that suits them. Proximity information is updated according to the user's current location when viewing the advertisement, in order to display more accurate and real-time information.

An *AdvertisementQueue* class has been implemented to store received advertisements. this is a first-in first-out type queue and is configured with a maximum size. If the maximum queue size has been reached, the oldest advertisements in the queue are replaced by newly arriving advertisements. The idea is that more recently advertisements are more likely to be relevant to the user's current situation/context.

Chapter 5

Evaluation

This chapter presents an evaluation of the context-aware advertising system presented in Chapters 3 and 4. The main focus of the evaluation is on a number of field experiments which were performed. The scenarios devised for the experiments attempt to replicate potential real-world usage scenarios of the system. The aim is to evaluate the system's ability to perform according to the design specification and also to identify potential issues and/or areas for potential improvement.

5.1 Experimental Scenarios

Two experimental scenarios have been devised. As mentioned, they are intended to model examples of potential typical real-life usage of the system. Multiple scenarios were used in an attempt to examine all of the key features of the system.

5.1.1 Experimental Setup

The following hardware equipment was used for the experiments:

- 2 x Dell Latitude D400 Laptops (Operating System : Microsoft Windows XP SP2, Wireless Card: Dell TrueMobile 1300 WLAN Mini-PC)

- 1 x Dell Inspiron E1405 Laptop (Operating System: Microsoft Windows XP SP2, Wireless Card: Dell Wireless 1390 WLAN Mini-Card)
- 2 x Magellan GPS 315 Receivers

The three laptops were configured to run in ad-hoc mode without encryption, enabling them to automatically create an ad-hoc network when within range of each other. The range of the wireless cards was found to vary considerably to anywhere between 50 and 250 metres. This appeared to depend on the factors such as the orientation and elevation of the laptops. The accuracy of the GPS receivers also varies but was found to be on average accurate to within 5-10 metres. The evaluation scenarios use larger proximities so GPS data precision within this range is not a major influencing factor in the experiments.

Two of the nodes used the GPS receiver to retrieve real-time location information, which were connected via the laptops' serial ports. The third node used a simulator program to provide its GPS location and remained stationary. Its location was determined using one of the GPS receivers prior to the commencement of the experiments and the simulator program was configured with this location. The experiments were performed in a large open park area to minimise any potential external interference and to simplify measuring out distances.

5.1.2 Scenario 1

Scenario Description:

A music venue is holding a live rock music concert and wants to advertise the gig in an attempt to attract potential customers. They create and schedule an advertisement to try and target potential interested consumers within close proximity. A new advertisement is created including details of the gig, artist and venue. A new sub-category of 'Music' named 'Rock' is created to provide a more specific categorisation of the advertisement. A proximity of 300 metres is specified for the advertisement to be delivered within as they believe this is the maximum distance a consumer would be prepared to walk to see the gig. The gig begins at 21:30 so the advertisement is scheduled to be delivered between 19:00 and 22:00 and on the Saturday only.

Two different consumers in the area have previously expressed an interest in receiving advertisements in the 'Music' category. The music venue is 70 metres from the Consumer 1 and 200 metres from Consumer 2. Consumer 1 is within wireless transmission range of the advertiser. Consumer 2 is not within direct range of the advertiser but is within range of Consumer 1. The consumer applications receive a category event with the new 'Rock' category which they are both automatically subscribed to. Soon afterwards, Consumer 1 receives an advertisement notifying them of the gig. Consumer 2 moves to within 150 metres of the advertiser, at which point it also receives the advertisement.

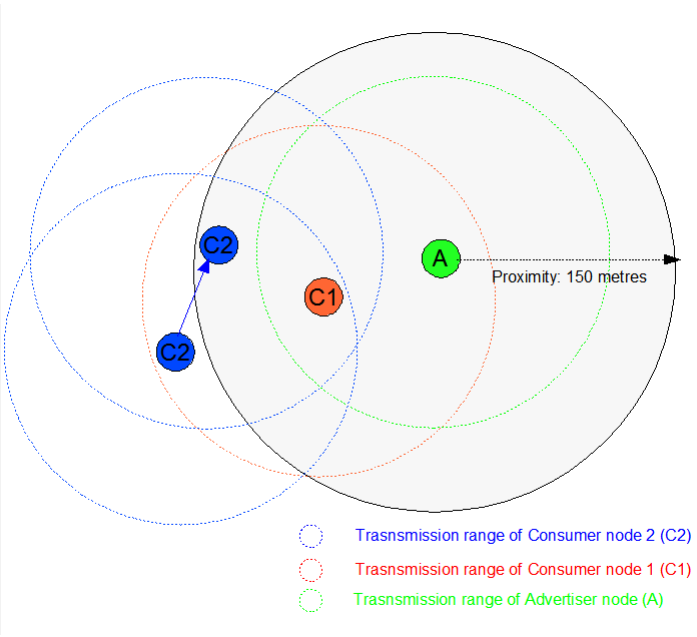

Figure 5.1: Scenario 1

Figure 5.1 above shows a graphical representation of this scenario. The Advertiser node (A) and Consumer 1 (C1) were stationary for the duration of the experiment. Consumer 2 (C2) moves from a starting position outside the advertiser's defined proximity to a position inside this proximity but outside the advertiser's transmission range. At this point, C1 is within transmission range of both the A and C2, thus allowing an ad-hoc network to be formed among the three nodes. This setup should enable the advertisement to be transmitted from A to C2 via multi-hop communication.

The event and announcement interval was configured to 10 seconds, meaning both ad-

vertisement and category events are sent every 10 seconds. Category event proximity is set to 2000 metres. Consumer frequency preference was set to 'always' while their proximity setting is set to 1000 metres. All nodes are configured to support multihop dissemination of messages, meaning they will forward messages they receive. Application data was written to log files to assist with analysis after the experiment. When moving, node C2 moved at an average speed of 2.5 km/hr. Each experimental test run lasted for a duration of 10-15 minutes. Most of the time is attributed to moving the nodes into position and ensuring nodes A and C2 were not within direct transmission range.

Results Summary:

Table 5.1 summarises the results for one of the completed test runs in this scenario. The advertisement was delivered to both C1 and C2 as expected at a proximity of 70.1 metres for C1 and 128.1 metres for C2. The table illustrates the number of events received by each node at the different layers of the system. It also shows which filters were applied.

	Advertiser (A)	Consumer 1 (C1)	Consumer 2 (C2)
PENCA Announcements received	n/a	960	301
PENCA Events Received	n/a	125	24
Category Events Sent/Received	167	42	10
Advertisement Events Sent/Received	167	19	2
Category Updates	1	1	1
Filter Updates	0	0	0
Consumer proximity filtered	n/a	0	0
Duplicate Ad Filter	n/a	18	1
Custom Filter	n/a	0	0
Advertisements Delivered	n/a	1 70.1 metres N 53 21 37.9 W 6 17 44.9	1 128.1 metres N 53 21 37.6 W 6 17 44.9

Table 5.1: Scenario 1 Results Summary

5.1.3 Scenario 2

Scenario Description:

A fashion store is running a promotion and trying to attract customers to its store. An advertisement is created using the context-aware advertising application to promote the store. The store has a large number of regular customers that have been issued with 'gold cards'. The manager wants to target the advertisements at these 'gold card' holders. An advertisement is created offering a discount to anyone that presents the advertisement in the store. The advertisement is categorised under the 'Fashion-Ladies' category. A custom filter is created and included with the advertisement. The filter requires the gold card code to be entered in order to receive the advertisement. The

manager gives the PDA running the advertiser application to one of the marketing staff to carry with them as they wander the streets.

A young shopper is in the area and has previously expressed an interest in receiving fashion advertisements. They receive the new 'gold card' filter and subscribe to it with their gold card code. Initially the consumer is 100 metres away from the advertiser. They both wander the streets to within 50 metres of each other. Shortly afterwards the consumer receives a notification of the advertisement.

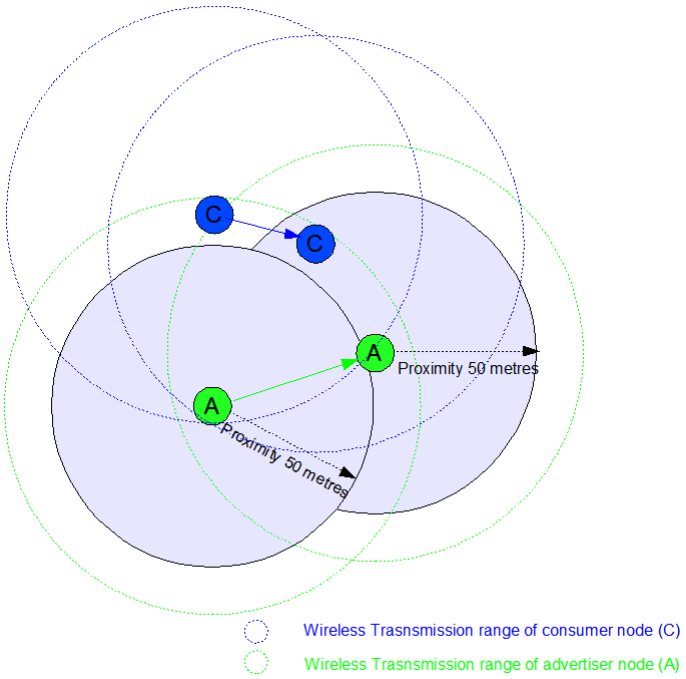

Figure 5.2: Scenario 2

Figure 5.2 graphically illustrates the setup for this second experiment. Both nodes were mobile in this scenario. At the starting positions, the nodes are within wireless transmission range of each other. Node C receives and subscribes to the newly received filter at this point. Node C is outside the proximity range (50 metres) defined for the advertisement so does not receive the advertisement at this point. Both nodes then move to a new position. Node C is then within 50 metres of the advertiser node (A). At this point, the advertisement is delivered to node C. The configuration information used was the same as that used for scenario 1, described in the previous section.

Results Summary:

Table 5.2 summarises the results from one of the test runs in these experiments. The advertisement was delivered as expected and was received at a proximity of 35.0 metres.

	Advertiser (A)	Consumer (C)
PENCA Announcements received	n/a	284
PENCA Events Received	n/a	78
Category Events Sent/Received	50	36
Advertisement Events Sent/Received	22	3
Category Updates	0	0
Filter Updates	1	1
Consumer proximity filtered	n/a	0
Duplicate Ad Filter	n/a	2
Custom Filter	n/a	0
Advertisements Delivered	n/a	1 35.0 metres N 53 21 7.12 W 6 18 49.5

Table 5.2: Scenario 2 Results Summary

5.1.4 Scenario Results Analysis

The results of the experimental scenarios demonstrate the overall functionality of the system. The features designed and implemented in the previous chapters produced expected results when tested in a real world type scenario. The system can therefore be said to perform according to the outlined requirements. Scenario 1 demonstrates the systems ability to reach multiple consumers without any prior knowledge of the consumers. The multi-cast transmission of events enables advertisers to reach multiple consumers with a single event. This solution is also very scalable in that the advertisers resource usage does not increase as more nodes enter the application space.

The multi-hop capabilities of the system are also demonstrated. Messages are retransmitted by application nodes, which enables events to be transmitted between disparate nodes that are not within direct transmission range. This provides potential for the application space to span relatively large geographical areas.

Full mobility capabilities are provided in that the system enables mobile advertisers and consumers to interact. The freedom for nodes to enter and exit the application space is another important feature. Nodes dynamically and spontaneously form the application space without any prior knowledge of each other.

The targeting and personalisation features of the system are realised through the combination of various filtering mechanisms. The experimental results illustrate the different filters applied and the levels at which they are applied. This combination of filters results in only the most relevant advertisements being delivered to consumers, resulting in potentially much greater consumer satisfaction than is possible with traditional advertising media.

An important aspect that is evident from the results of the experiments is the number of messages received by consumer nodes that are not ultimately used or delivered to the consumer. This can be seen when comparing the number of PENCA events received with the number of CAAF event received and then the number of advertisements actually delivered. While the different levels of filtering result in only relevant and fresh information being delivered to the end user, the consumer device is still required to process a large amount of what turns out to be redundant information.

It is important to attempt to filter any messages at the earliest or lowest layer possible in order to minimise the use of resources. The fact that more messages are received by nodes than was initially sent out by the advertiser can be attributed to multi-hop message forwarding that occurs. The experiments performed used a small number of nodes and a relatively small number of messages. A further evaluation could focus on analysing the systems performance in a larger application space with a large number of nodes and with a large number of messages being transmitted. There is potential in this type of environment for a node to be required to deal with a substantial amount of message processing, much of which may be redundant. This is of particular concern in mobile resource-limited environments. The total amount of redundant processing performed is dependent on the application environment. For instance, the more mobile the

application nodes are, the more likely they are to encounter 'fresh' and potentially relevant messages. There is a trade-off when configuring parameters such as event interval between maximising the penetration and number of nodes advertisement messages reach, and minimising the amount of processing performed by application nodes.

5.2 Security Evaluation

As part of the evaluation, a number of potential security threats or issues have been identified: The widely-used CIA (Confidentiality, Integrity and Availability) triad information assurance model has been used as part of this evaluation. The security analysis is summarised under these headings:

- Confidentiality: refers to the data privacy and the idea that only intended and authorised recipients read the data. All information in this system which is transmitted is essentially broadcast for public consumption. The nature of advertisement data therefore means that confidentiality is not required. Potentially sensitive consumer data is not transmitted over the network and is only used locally.
- Integrity: is the assurance that the information has not been altered in transmission. The implemented system identifies the source of messages by its advertiser id. This id is used in the feedback mechanism. The system is open in that any advertiser can create any ad content it wants. The advertiser id is currently configured in a properties file which is open to tampering by attackers. In order to provide system integrity, some type of licensing mechanism could be implemented where advertisers are issued with a key to enable them to publish advertisements. Consumers would only then consume advertisements signed with a valid key. However, consumers would need to be configured with all necessary information from the offset as there is no infrastructure or central component with which it could communicate to retrieve this information.
- Availability: is the idea that information or resources are available when required. Since the application is formed in an ad-hoc manner without any infrastructure, it is entirely dependent on other application nodes being nearby in order to operate. This is outside the control of the system. There is potential for Denial of Service

attacks in which an attacker (e.g. a rival advertising service provider) could overload the application with broadcast messages. The multi-hop forwarding feature of the system means that the effect of any such attack would in fact be magnified, thus making this type of attack easier for the attacker.

- Non-Repudiation: refers to the assurance that the sender of data is provided with proof of delivery and the recipient is provided with proof of the sender's identity. The nature of the multi-cast broadcasting of messages implemented in the system means that proof of delivery is not provided. The sender also does not know the message's recipients in advance. Again, the sender information is included with each message but a robust scheme would be required in order to provide non-repudiation.

As is evident from the security evaluation, the system is quite open to a number of security threats. Spammers would be quite likely to target this type of application as it provides an ideal medium for SPAM advertisements. There is also potential for misinformation in that the sender of a message cannot currently be guaranteed. The nature of the application means that sensitive data is not being transmitted over the network. However, locally stored data may potentially be of a sensitive nature (e.g. user preferences, unpublished advertising campaigns). Some type of encryption or protection mechanism would need to be implemented in order to protect this data. The fact that application nodes operate in ad-hoc mode and actively search for other random nodes to connect to also provides a significant security concern. The network is essentially open to any would-be attacker to join. Generally, this type of open unsecured network environment is not reccommended from a security perspective.

This project focuses more developing a prototype system to demonstrate the core functionality possible rather than creating a commercially deployable system. However it is apparent that there are a number of significant security considerations that would need to be addressed if a commercially deployable system was to be created.

Chapter 6

Conclusion

This chapter presents a conclusion drawn from the information presented in the exploration, design, implementation and evaluation of this dissertation. The primary aim was to investigate the potential for a context-aware advertising service in a mobile ad-hoc collaborative environment and to do so by building on previous research to develop proximity and subject filtered event-based middleware.

The context-aware advertising system presented effectively demonstrates the potential for collaborative advertising applications in ad-hoc networking environments. The lack of a reliance on any infrastructure is a major positive feature of the system which provides great flexibility and mobility. This feature also distinguishes from many of the currently available mobile advertising services. The combination of different filtering mechanisms provides advertisers with the capabilities to target advertisements based on user's current context. This is a very powerful feature for advertisers and is something that is not currently possible with existing advertising media. The highly dynamic nature of application nodes leads to the dynamic acquisition of context, and subsequently a system that is highly adaptive and capable of tailoring its information to a user's current context.

The system enables advertisers to target advertisements at specific times, locations & demographics. At the same time, consumers receive more personalised & relevant advertisements, resulting in a system that is less intrusive and therefore more likely to gain consumer acceptance.

There are some potential issues with the system as discussed in the evaluation section. The overall efficiency and use of resources may be an issue and requires further investigation. As discussed previously, efficient use of resources is of particular importance in mobile environments due to te fact that mobile devices generally have significantly limited computing resources available. A number of security issues have also been identified which would need to be addressed in order to commercialise this type of the system.

6.1 Future Work

A number of potential areas for future work and research have been identified and are summarised as follows:

- A further performance and efficiency evaluation could be performed as discussed in the evaluation section. Included in this evaluation could be an investigation into providing filtering on the producer side. Conducting a survey with consumers and advertisers in order to gauge and evaluate the relevance of advertisements delivered by the system compared to traditional forms of media.
- A further area for future work could be to attempt to reduce the level of user interaction required by the consumer and increase the system's ability to automatically learn the consumer's preferences. Consumers have been found to prefer this type of interaction despite the increased lack of control it introduces.
- The end user applications developed in this project were not designed specifically for devices such as PDAs. Further applications specific to these types of devices could be developed, while also investigating the potential for delivering different types of advertising content. Additional features that could be developed include providing consumer 'pull' services, where a consumer could request a certain type of advertisement (e.g. give me a list of nearby taxis). Also, creating a graphical visualisation of a map interface could be useful from an consumer end-user perspective.
- The implemented system uses location data based on a GPS sensor. A significant limiting factor with this is the fact that it only operates outdoors. Future work

could focus on developing additional location services that provide location information in an indoor environment.

- If the system was to be commercialised, there are a number issues that would require further work. The security considerations discussed in the evaluation section would need to be addressed. In particular, developing mechanisms to counter the potential threat of spam advertisements. Also, because there is a lack of reliance upon any infrastructure, there is no simple way to introduce tariffs for the service. Investigating different possibilities for metering the use of the service and subsequently charging advertisers could be another area for investigation.

Bibliography

[1] R. Meier, *Event-Based Middleware for Collaborative Ad Hoc Applications*. PhD thesis, Department of Computer Science, University of Dublin, Trinity College, Sept. 2003.

[2] K. Chatar, "Lightweight middleware for mobile java events.".

[3] A. S. E. Catherine Soanes (Editor), *Oxford English Dictionary*. Oxford University Press, 2005.

[4] "Uk internet advertising statistics 2007 report," 2007.

[5] eMarketer, "Us mobile advertising revenues, 2006 and 2011," 2007.

[6] A. Dey, "Understanding and using context," 2001.

[7] "Mobile ad-hoc network. wikipedia."

[8] N. Hristova and G. M. P. O'Hare, "Ad-me: Wireless advertising adapted to the user location, device and emotions," in *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, (Washington, DC, USA), p. 90285.3, IEEE Computer Society, 2004.

[9] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *CHI*, pp. 434–441, 1999.

[10] R. Meier and V. Cahill, "Exploiting proximity in event-based middleware for collaborative mobile applications," 2003.

[11] P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiqitous computing* (H.-W. Gellerson, ed.), no. 1707 in Lecture Notes in Computer Science, pp. 304–7, Springer, September 1999.

[12] G. H. M. Ebling and H. Lei, "Issues for context services for pervasive computing," in *In Proc. of the Advanced Workshop on Middleware for Mobile Computing*, 2001.

[13] M. Korkea-Aho, "Context-aware applications survey," tech. rep., Helsinki University of Technology, Department of Computer Science, April 2000.

[14] P. Moore and B. Hu, "Entity profiling: Context with ontology," in *Pervasive Computing and Applications, 2006 1st International Symposium on*, pp. 172–177, August 2006.

[15] L. Barkhuus and A. Dey, "Is context-aware computing taking control away from the user? three levels of interactivity examined," 2003.

[16] K. D. O. Drogehorn, B. Wust, "Personalised applications and services for a mobile user," in *Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings*, pp. 473– 479, IEEE Computer Society, 2005.

[17] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: a mobile context-aware tour guide," *Wirel. Netw.*, vol. 3, pp. 421–433, October 1997.

[18] L. Aalto, N. Göthlin, J. Korhonen, and T. Ojala, "Bluetooth and wap push based location-aware mobile advertising system," in *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, (New York, NY, USA), pp. 49–58, ACM Press, 2004.

[19] R. Bulander, M. Decker, B. Kolmel, and G. Schiefer, "Enabling personalized and context sensitive mobile advertising while guaranteeing data protection," in *Proceedings of EURO mGOV 2005* (I. Kushchu and M. H. Kuscu, eds.), (Brighton, UK), pp. 445–454, Mobile Government Consortium International LLC, 2005.

[20] O. Ratsimor, T. Finin, A. Joshi, and Y. Yesha, "encentive: a framework for intelligent marketing in mobile peer-to-peer environments," in *ICEC '03: Proceedings of the 5th international conference on Electronic commerce*, (New York, NY, USA), pp. 87–94, ACM Press, 2003.

[21] I. Podnar, M. Hauswirth, and M. Jazayeri, "Mobile push: Delivering content to mobile users," in *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, (Washington, DC, USA), pp. 563–570, IEEE Computer Society, 2002.

[22] H. Interactive, “Mobile advertising and marketing usa conference,” 2007.

[23] M. Musolesi, C. Mascolo, and S. Hailes, “Adapting asynchronous messaging middleware to ad hoc networking,” in *MPAC ’04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, (New York, NY, USA), pp. 121–126, ACM Press, 2004.

[24] Y. Huang and H. Garcia-Molina, “Publish/subscribe in a mobile environment,” *Wirel. Netw.*, vol. 10, no. 6, pp. 643–652, 2004.

[25] S. Oaks and H. Wong, *Jini in a nutshell: a desktop quick reference*. Sebastopol, CA, USA: O’Reilly & Associates, Inc., 2000.

[26] “Jini.org the community resource for jini technology. http://jini.org.”

[27] *Jini Technology Architectural*, 1999.

[28] *Jini Distributed Events Specification*.

[29] T. White, “Scheduling recurring tasks in java applications,” 2003.

[30] *Swing Wizard Framework. https://wizard-framework.dev.java.net/*.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley Professional, 1995.

[32] “Log4j. http://logging.apache.org/log4j/1.2/index.html.”

[33] “Dom4j. http://www.dom4j.org/.”

[34] “Jcalendar. https://jcalendar.dev.java.net/.”