CHANNEL SELECTION ALGORITHM FOR SOFTWARE DEFINED RADIO BASED COGNITIVE RADIOS

by

Adem Zümbül

B.S., Computer Engineering, Marmara University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2009

CHANNEL SELECTION ALGORITHM FOR SOFTWARE DEFINED RADIO BASED COGNITIVE RADIOS

APPROVED BY:

Assoc. Prof. Tuna Tuğcu (Thesis Supervisor)	.....
Assoc. Prof. Fatih Alagöz	.....
Dr. Hamza Özer	.....
Assist. Prof. Gökhan Yavuz	.....

DATE OF APPROVAL: 15.06.2009

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Assoc. Prof. Tuna Tuğcu for his kind support, assistance, and patience during my thesis. Without his guidance, it would be impossible for me to finish this thesis.

I would like to thank all TÜBİTAK UEKAE team whose endless support made this work possible. I have enjoyed getting to know each of them and appreciate not only the technical knowledge they have imparted to me, but also their friendship.

I would also like to thank my parents, Recep and Şükriye Zümbül, and my brothers, Osman and İsmail Zümbül, for their tireless support of my ever-changing passions and goals throughout the years. They have encouraged me, picked me up when I am down, and kept me anchored through this entire process. They have supported me every step of the way and they have truly made this all worthwhile.

I gratefully acknowledge the help of the people in proof-reading this document, in some cases several times.

Thank you for believing in me.

ABSTRACT

CHANNEL SELECTION ALGORITHM FOR SOFTWARE DEFINED RADIO BASED COGNITIVE RADIOS

Radio spectrum is a finite resource and effective utilization of it in wireless networks is a key challenge as the number of users increase. Current wireless networks are expected to fail to satisfy increasing user demands due to inefficient spectrum management resulting from the fixed assignment policy in which each wireless network has its own running parameters. Current hardware-based technology does not allow dynamic usage and is very cumbersome.

Dynamic spectrum allocation, which can be achieved by cognitive radio (CR) technology that is based on software defined radio (SDR) architecture is a promising solution for efficient spectrum utilization. CR is an intelligent device that automatically senses, recognizes, and makes wise use of idle parts of the spectrum dynamically. CR achieves dynamism by making handoffs to underutilized bands. Handoff is an expensive operation. Because, it requires suspending an ongoing communication, searching and selecting a new channel, and reconfiguring CR to switch that channel. Also, all of these operations should be performed in the shortest time to avoid communication problems. Decreasing number of handoffs is a key challenge for efficient operation of CR.

Initial studies for handoff are based on sense-and-react approach where handoff is made solely based on current spectrum observations. This approach may lead to possible communication failures because users cannot foresee future channel status. Other handoff algorithms mostly focus on determining the handoff time, increasing bandwidth usage, achieving faster channel discovery, and minimizing disturbance to primary users but do not answer the question of "how to minimize number of handoffs by considering user behavior?" In addition, existing channel selection algorithms

provide only software simulation or hardware testbed results and neglect the software design and implementation details of their approach on a real SDR. In this thesis, two channel selection algorithms are proposed, and an infrastructure based SDR implementation is provided to answer these questions. Proposed channel selection algorithms aim to learn user behavior and use channel utilization histories for predicting the new candidate channel for handoff. In the implementation section of the thesis, software design challenges of a SDR based CR are discussed and several software design patterns are proposed, the layers of the software and components in each layer is explained, the results from a software engineering point of view are examined and finally, the lessons learned and troubles encountered during the implementation are presented.

ÖZET

YAZILIM TANIMLI TELSİZ TABANLI AKILLI RADYOLAR İÇİN KANAL SEÇME ALGORİTMASI

Radyo spektrumu sınırlı bir kaynaktır ve kablosuz ağlardaki artan kullanıcı sayısı ile birlikte radyo spektrumunun etkili bir şekilde kullanılması kilit bir problem olmuştur. Artan bant genişliği kullanımı göz önüne alındığında var olan kablosuz ağların yetersiz kalacağı sonucuna varılabilir. Bu durumun temel nedeni her kablosuz ağın kendi çalışma parametrelerine sahip olduğu sabit spektrum atama politikasından kaynaklanan spektrum yönetimindeki verimsizliktir. Günümüzde kullanılan donanım tabanlı teknoloji dinamik kullanıma izin vermez ve oldukça hantaldır.

Dinamik spektrum tahsisi, verimli spektrum kullanımı için umut verici bir çözümdür ve gerçeklenmesi ancak yazılım tabanlı telsiz (YTT) mimarisine dayanan akıllı radyo (AR) teknolojisi ile mümkündür. Akıllı radyo, spektrumun boş kısımlarını otomatik olarak algılayan ve dinamik olarak kullanılmasını sağlayan akıllı bir cihazdır. Akıllı radyo bu dinamizmi çalışma esnasında kanal değişimleri yaparak sağlar. Ancak, kanal değiştirme pahalı bir operasyondur. Çünkü devam eden iletişimin bekletilmesini, yeni bir kanal seçilmesini ve AR'nun seçilen kanala göre yeniden ayarlanmasını gerektirir. Ayrıca, iletişim problemlerinin engellenmesi için bütün bu operasyonlar en kısa zamanda gerçekleştirilmelidir. AR'nun etkili bir biçimde çalışabilmesi için kanal değiştirme sayısının azaltılması çok önemlidir.

Kanal değiştirmeye yönelik ilk çalışmalar hali hazırdaki spektrum gözlemlerine dayalı algıla-ve-tepki ver yaklaşımına dayanır. Bu yaklaşım kullanıcıların gelecekteki kanal durumu hakkında öngörüde bulunamamaları dolayısı ile iletişim problemlerine neden olabilir. Diğer kanal değiştirme algoritmaları ise genellikle kanal değiştirme zamanının belirlenmesi, bant genişliği kullanımının arttırılması, daha hızlı kanal keşfetme

ve birincil kullanıcılara verilen zararın en aza indirilmesine odaklanmaktadır ancak "Kullanıcı davranışı göz önüne alarak kanal değiştirme sayısı nasıl azaltılabilir?" sorusuna cevap vermemektedir. Ayrıca, var olan kanal seçme algoritmaları sadece yazılım simülasyonu ya da donanım test ortamı sonuçları sunmaktadır ve sunulan yaklaşımların gerçek bir YTT üzerinde yazılımsal olarak nasıl gerçekleneceğine değinmemektedirler. Bu tezde, belirtilen eksikliklere cevap verebilmek için iki adet kanal seçme algoritması önerilmiş ve altyapı tabanlı bir YTT uygulaması gerçeklenmiştir. Önerilen kanal seçme algoritmaları kullanıcı davranışlarını ve kanal kullanım geçmişlerini öğrenmeyi ve bunlardan faydalanarak seçilecek yeni kanalı belirlemeyi hedeflemektedir. Tezin gerçekleme bölümünde ise, YTT temelli bir AR'nun yazılımsal tasarım problemleri irdelenmiş ve bazı yazılım tasarım desenleri önerilmiştir. Ayrıca, yazılım katmanları ve her katmandaki bileşenler açıklanmış, sonuçlar yazılım mühendisliği bakış açısı ile değerlendirilmiş ve son olarak öğrenilen dersler ve gerçekleme sırasında karşılaşılan sorunlar sunulmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

API	Application Program Interface
CF	Core Framework
COTS	Commercial Off-the-Shelf
CORBA	Common Object Request Broker Architecture
CR	Cognitive Radio
CRC	Canadian Research Center
DSP	Digital Signal Processor
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor
IDE	Integrated Development Environment
IDL	Interface Definition Language
JPO	Joint Program Office
JTRS	Joint Tactical Radio System
OMG	Object Management Group
OOP	Object Oriented Programming
OSSIE	Open Source SCA Implementation::Embedded
PRF	Properties Descriptor
SCA	Software Communications Architecture
SCARI	SCA Reference Implementation
SCD	Software Component Descriptor
SDR	Software Defined Radio
SPD	Software Package Descriptor
UML	Unified Modeling Language
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WF	Waveform
XML	Extensible Markup Language

1. INTRODUCTION

In this chapter, we define the problem being investigated in this thesis, list our contributions and present the organization of the thesis.

1.1. The Problem

Spectrum management in a wireless network becomes a key challenge as the number of users increase. Considering the current trend of increasing bandwidth usage, we can conclude that current wireless networks will fail to satisfy user demands. The main cause is inefficiency in spectrum management due to the fixed assignment policy in which each wireless network has its own running parameters such as frequency and modulation type that can not be changed during operation [1, 2]. Current hardware-based technology does not allow dynamic usage and is very cumbersome.

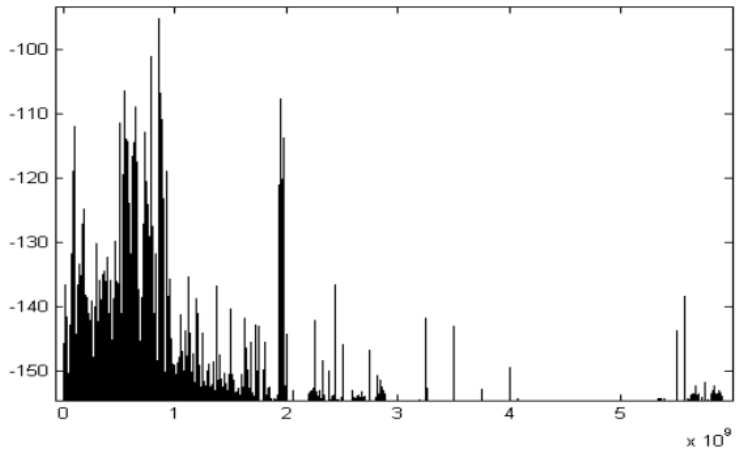

Figure 1.1. Measurement of 0-6 GHz spectrum utilization at Berkeley Wireless Research Center [3]

A typical spectrum utilization graph showing the measurements of the 0-6 GHz band at Berkeley Wireless Research Center [3] is shown in figure 1.1. Although the spectrum is wide enough, it is not used homogenously, which constitutes the main source of the inefficiency problem. However, this problem also represents the solution itself. That is, some portion of the spectrum is heavily used while the rest of the

assigned spectrum is used sparsely. Some frequency bands suffer from service quality due to heavy load while low utilization of some other bands decreases spectrum efficiency. Current architectures of the wireless handsets and wireless stations do not offer any solution to solve these problems because the technology behind most of the current handsets is not capable of operating in a dynamic manner. They have fixed and predefined software and hardware that do not adapt to the dynamic structure of their operating environment. In order to achieve dynamism, a relatively new approach to traditional radios emerges: Software Defined Radio (SDR) [4, 5].

Assuming that an ideal SDR can offer the maximum dynamism to a radio, a new problem arises: determining the available spectrum and tuning to most appropriate place in the large spectrum area by sensing the internal state of the radio and the surrounding communication environment. The radio should provide ability to support dynamic reconfiguration. The ability of the radio to reconfigure itself at runtime allows it to use the sparse spectrum when needed. It can be defined as an opportunistic behavior. Current solutions to this problem introduce a new concept to the radio world, Cognitive Radio (CR) [6, 7].

CR offers dynamic spectrum access in order to solve inefficiencies in spectrum utilization techniques. It aims to use shared spectrum without causing significant harmful interference with licensed users (primary users). CR autonomously coordinate the usage of the spectrum by observing RF environment when it is unused by the others. Such unused radio spectrum is called “spectrum opportunity” or “white space” [8]. CR monitors the existence of other users and switches its communications to other channels dynamically, which is called “handoff” [9, 10].

Handoff is an expensive operation since, it requires suspending an ongoing communication, selecting a new channel, and reconfiguring CR to switch that channel. Also, all of these operations should be performed in the shortest time to avoid communication problems. Perfect handoff is a key challenge for efficient operation of CR. It includes several difficult questions to answer. Some of them can be listed as follows [1]:

- **Channel selection:** The hardest question for a CR is deciding the channel to switch the communication during handoff. CR should be able to select a channel where it can continue communication for the longest period of time without needing a new handoff.
- **Handoff timing:** Determining the time for handoff is another hard question for a CR. The time of handoff is critical and should be decided precisely to avoid communication failures. Perfect time for a handoff should be decided by predicting or sensing the existence of a primary user.
- **Sensing Errors:** If the handoff is determined by sensing, CR should be able to sense existence other users and their types (primary user or secondary user) perfectly in order to avoid interference.

All of these facts require that the number of handoff should be minimized in ideal CR. In this thesis, we focus on channel selection for handoff optimization. We propose two channel selection algorithms and evaluate them for different scenarios. We also implement an SCA based SDR to realize our algorithms.

1.2. Definitions

1.2.1. Software Defined Radio (SDR)

SDR is a system in which running parameters and software can be dynamically adjusted. Ideally, digitalization in a software radio starts right after the antenna. Ideal SDRs are the most flexible communication devices that can operate as different radios by installing new applications. The hardware of a SDR is controlled by the running software on it so that it can tune to any frequency band and receive any modulation across a large frequency spectrum by the help of programmable hardware. If the software on a SDR is changed, the capability of the radio is also changed as long as the hardware allows. The logic behind a software radio is based on replacing the radio hardware components (analog to digital converter (ADC), digital to analog converter (DAC), modulator, and filter) with the equivalent piece of software. In order to handle performance issues related to fast signal processing, FPGA (Field Programmable Gate

Array) and DSP (Digital Signal Processor) based processing hardware is used instead of GPP (General Purpose Processor).

The opportunities that SDR offers make it a candidate platform to solve the problems related to dynamic spectrum assignment. Capabilities of a SDR allow the handset to operate in a heterogeneous wireless network. In other words, an ideal SDR handset can dynamically change its running software and can tune to available frequency and modulation type at runtime. However, there are many open issues in SDR. Firstly, SDR hardware is not standardized causing SDR software to be platform dependent. Also, the operating environment (operating system, drivers, and board support packages) of a SDR may vary in most SDR platforms. Implementing platform independent SDR software and porting one SDR application to another SDR platform is a key issue.

1.2.2. Cognitive Radio (CR)

CR is a wireless communication system that is aware of its running environment and is able to change its transmission and reception parameters to communicate efficiently in its wireless network. This alteration of the running parameters is based on the active monitoring of several factors in the external and the internal radio environment. These are typically radio frequency spectrum, user behavior, and network state. In addition, depending on the resource management algorithm, a CR may also monitor other parameters such as cost, network usage costs, location based parameters, policy based parameters, weather conditions, distance to wireless stations, and so on. However, this monitoring job is not an easy task and requires a lot of hardware and software resources as well as processing power. Such issues are beyond the scope of this thesis, but our work provides the necessary basis for research on CR.

1.2.3. Waveform

A waveform is the representation of a signal that includes the frequency, modulation type, message format, and/or transmission system. In general usage, the term

waveform refers to a known set of characteristics, for example, frequency bands (VHF, HF, UHF, etc.), modulation techniques (FM, AM, etc.), message standards such as Link 16, and transmission systems (SINCGARS, EPLRS, HAVEQUICK, etc.). In this thesis, the term waveform is used to describe the entire set of radio functions that occur from the user input to the RF output, and vice versa. In other words, it can be explained as the set of transformations applied to information to be transmitted and the corresponding set of transformations to convert received signals back to their information content.

1.3. Contribution of This Thesis

The main contributions of this thesis can be listed as follows:

- Two dynamic channel selection algorithms are proposed for the CR to minimize average handoff per call.
- A channel generator and a CR simulator tool are implemented to evaluate proposed algorithms on different scenarios.
- In addition to simulators, proposed algorithms are also realized by implementing a real SCA compliant SDR application (Waveform) with C++ programming language. The implementation is also evaluated for different SDR metrics including reconfigurability, portability, reusability, scalability, interoperability, upgradability, and affordability.
- Our SDR implementation provides reusable and open-source software components that can be used to build up new applications and our design can guide the development of future radio systems. It is currently being considered by the SDRForum as a reference implementation.
- CR requirements are analyzed and their mapping to the software architecture of SDR is investigated.
- Design challenges of the software layers (operating system, middleware (CORBA), configuration management system (XML), core framework, and waveform) of a SDR that allows the radio to operate in a dynamic manner are discussed. It includes object oriented design of the SDR platform and its running software. Current solutions are compared in terms of pros and cons and also extended by applying software design patterns.
- Finally, our results are examined from a software engineering point of view and the lessons learned, troubles encountered, design steps, and the tools that have been used during the implementation are presented.

1.4. Thesis Organization

This thesis is presented in six chapters and is organized as follows:

Chapter 2 presents the literature survey about spectrum management and channel selection, overview of different SDR software architectures, design patterns, middleware, and configuration management protocols.

Chapter 3 explains our proposed channel selection algorithms for the CR to minimize the number of handoffs during its operation.

Chapter 4 explains our SDR design and its layers. It presents the design and implementation steps of the waveform that has been implemented in our platform. It also proposes some design patterns to extend our architecture.

Chapter 5 presents our evaluation results for the proposed channel selection algorithms on different scenarios. It also discusses our SDR implementation and summarizes the lessons learned and the difficulties encountered while implementing and testing the components.

Chapter 6 presents our conclusions and summarizes our observations. We then discuss possible future areas of work.

2. BACKGROUND

In this chapter, literature survey and background information is presented to explain the terms and concepts that are discussed in later chapters.

2.1. Previous Work in the Literature

This section presents literature survey on spectrum management techniques, dynamic channel selection algorithms, existing software architectures for SDR, and ongoing SDR projects.

2.1.1. Spectrum Management Techniques

In the literature, radio is defined as the technology for transmitting or receiving electromagnetic radiation over wireless medium to facilitate transfer of information [11]. The working principle behind a radio depends on the modulation of electromagnetic waves with frequencies below the visible light. Each wireless device sends or receives data in a predefined frequency in the spectrum. The receiver and the transmitter must be aware of each other's frequencies to maintain successful communication. If a radio transmits on a busy frequency without checking its availability the signals interfere and communication fails. In order to avoid interference, the frequency bands of wireless telecommunication system are fixed by assignment-in-advance which is called fixed spectrum allocation [1, 12].

At first glance, fixed spectrum allocation seems to be the ideal solution to manage the frequency spectrum for a finite number of users. However, it leads to poor utilization of the spectrum because of pre-assigning the spectrum without considering utilization. All parts of the spectrum are not used in a homogenous manner. Some frequency bands, especially below 3 GHz, are heavily used whereas some frequency bands are idle or under utilized [1]. For example, experiments in [13] shows that there is only about 5.2% of the spectrum below 3GHz is actually in use.

On the other hand, as time goes by, more and more devices go wireless such as laptops, cell phones, sensors, printers and even the cameras. All of these gadgets have to share a finite amount of radio spectrum but there is just no more space in the part of the radio spectrum designated for their fixed frequencies. This limitation constitutes a serious bottleneck that makes radios stop working.

In order to solve this bottleneck, frequency assignments have to be done in a more flexible way using dynamic spectrum allocation [14, 15, 16]. Dynamic spectrum allocation improves the overall spectrum efficiency by allowing wireless devices to use idle spectrum when needed. For example, in [17] a technique to utilize unused analog TV bands by other devices is introduced. In addition, dynamic spectrum allocation ensures that the available spectrum matches the user needs and provides more flexibility in emergency and natural disaster cases.

In the literature, frequency management is usually regarded as the basic spectrum management methodology [18, 19, 20, 21, 22]. However, it is not the only solution to achieve better spectrum efficiency. There exist some studies that explore more ways to adaptively access all dimensions associated with electromagnetic spectrum including time, space (both location and signal directionality), power, coding, polarization, and other signal features [23, 24]. For example, [24] defines the "virtual cube" concept which is shown in figure 4.8. According to this definition spectrum is investigated in three dimensions including frequency, time, and power-code. This model allows defining heterogeneous access types in existing networks based on a generic spectrum unit.

Determining a common spectrum unit is crucial for efficient utilization of the wireless spectrum. In the literature, almost all spectrum sharing techniques, consider channel as the basic spectrum unit for operation. A channel can be defined as the basic unit of the large spectrum. Although, all parts of the large spectrum is not constant because of the operating frequency, it is usually assumed to provide the same bandwidth as other channels [25, 26, 27, 28, 29].

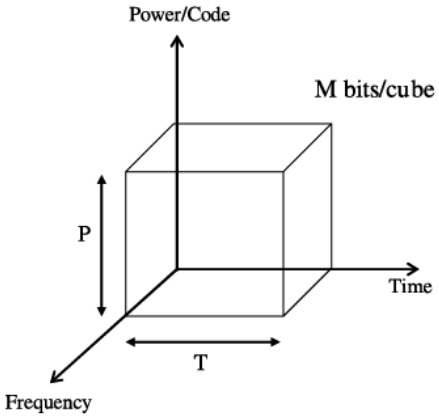

Figure 2.1. Virtual Cube

The existing solutions for spectrum sharing can be mainly classified in three categories:

- **According to architecture**
  - **Centralized architecture:** In centralized solutions, there exist a central authority which controls the spectrum allocation and access procedures [30, 31, 32]. In this approach central authority may collect spectrum information from operating devices in its network or it may sense all the spectrum itself.
  - **Distributed architecture:** In the literature, distributed solutions are usually proposed for scenarios where construction of an infrastructure is difficult or not preferable [25, 26, 19, 22, 33, 29]. In this approach, each node is responsible for its own spectrum management and interference detection to other units.
- **According to spectrum allocation behavior**
  - **Cooperative:** In cooperative approach, each node senses the spectrum and shares this information with other nodes [30, 25, 26, 19, 33]. Therefore, operating devices in the environment do not have to sense all the spectrum by itself. Cooperative approach assumes that there exists a common protocol between devices to share spectrum information. Each centralized architecture can be considered to be cooperative, however each cooperative approach does not have to include a central authority.
  - **Non-Cooperative:** In this approach, all devices are independent and does

not share spectrum information with each other [22, 34, 29]. Non-cooperative solutions are applicable to heterogeneous scenarios where operating devices does not have common spectrum information sharing protocol or there is no central management authority.

- **According to spectrum access technique**
  - **Overlay:** Overlay is a spectrum access technique for minimizing interference. In this approach, secondary users detect spectrum holes and use them opportunistically when the primary users are not active [35, 30, 25, 19, 22, 33, 29].
  - **Underlay:** In this technique, secondary users operate below the noise threshold of the high power licensed devices to prevent interference. In this type of communication secondary users are allowed for only very short-range devices. This technique requires advanced spread spectrum techniques and can utilize increased bandwidth in comparison to overlay techniques [26].

CR aims to provide opportunistic access to the licensed spectrum as secondary users. Spectrum sharing techniques of CR allow multiple radios to operate in the same environment without causing interference with each other. In other words, it provides capabilities to achieve concurrent communication in overlapping locations. Therefore, spectrum sharing is an important research topic. CR provides spectrum sharing by observing the usage of the RF spectrum. If CR detects a primary user during a call, it suspends the ongoing call, selects a new location in the spectrum, and switches the communication to that location. This operation is called "handoff".

Handoff is the key enabler operation of CR to achieve dynamic spectrum access. From a resource management perspective, selecting the network that best suites the user's demands and deciding when is the optimum time to handoff are challenging issues.

In the literature, studies on handoff are characterized into vertical handoff and horizontal handoff [36, 37, 38] in terms of involved technologies. A vertical handoff is a handoff between two different wireless access technologies. For example, when a mobile

terminal switches between an 802.11g network into a GPRS network, the handoff would be considered a vertical handoff. It is applicable in heterogeneous wireless networks and successful vertical handoff is a challenging question since it requires the decision maker to be aware of many parameters such as existing wireless access technologies at the location of the mobile terminal, spectrum utilization information of the users in each technologies, policies and regulations of each network, capabilities of the mobile terminals involving in handoff operation, and so on. Therefore, much of the studies on vertical handoff focuses on maintaining seamless handoffs that also preserve QoS.

In [38], Abd-Elhamid *et al.*, list the elements to establish a vertical handoff framework and investigate vertical handoffs as a resource management tool. In [39], Chakravorty *et al.* report some measurements on vertical handoffs between GPRS Cellular and WLAN hot-spots. They show that vertical handoff between these technologies causes major performance problems on TCP in IPv6 setting. Wu *et al.* propose an SIP based-solution for smooth vertical handoff between WWAN and WLAN technologies in [39]. In [40], Bernaschi *et al.* presents a setting to investigate IPv6/IPv4 interoperability in vertical handoffs.

A horizontal handoff is a handoff between the same network access technologies. For example, a mobile device makes horizontal handoff when moving in and out of various 802.11b network domains. In horizontal handoff, connection may be disrupted by device mobility. Horizontal handoff is relatively easier than vertical handoff since the involved technologies are homogenous, however vertical handoff in conjunction with horizontal handoff can provide more flexible operation for the CR.

From handoff management perspective, studies in the literature are investigated in three categories: mobile-controlled handoff (MCHO), network-controlled handoff (NCHO), and mobile-assisted handoff (MAHO) [41]. In MCHO, decision maker is the mobile terminal itself. An example of MCHO can be given as IEEE 802.11 WLAN networks. In NCHO strategy, handoff decision is directly given by the central spectrum management authority. As an example, cellular voice networks use NCHO for handoff decision. In MAHO, both mobile terminal and the central authority are involved in

handoff decision. MAHO has been adopted in the current WWAN networks such as GPRS, where the MH measures the signal of the surrounding base stations and the network then uses this information for handoff decision.

Initial studies for handoff proposes a sense-and-react approach [42, 43, 44]. In this approach, handoff is decided directly by considering threshold comparison of measured spectrum metrics. This category can also be referred as reactive approach. Studies in this category, uses traditional signal intelligence algorithms for handoff decision. These algorithms generally consist of three main stages:

- **Spectrum sensing:** In this stage, signals are collected from the RF environment. Radios monitor spectrum channels through individual or collaborative sensing [45, 46, 47, 48, 49, 50].
- **Signal analysis:** This stage consists of analyzing the collected signals for deriving signal metrics. The most common metrics are received signal strength (RSS), carrier-to-interference ratio (CIR), signal-to-interference ratio (SIR), and bit error rate (BER) [41].
- **Decision:** In decision stage, measured values from the environment are compared with the pre-defined threshold values. As a result of that comparison, handoff decision is given.

Reactive approach is widely used in cellular networks [36]. Since sense-and-react approach is independent of the wireless access technology, it can also be applied to heterogeneous wireless networks. This is a result of the fact that each wireless network uses a specific signal (beacon, BCCH, or reference channel) with a constant transmit power which can be used for deciding the time for triggering handoff.

Sense-and-react approach is based on current spectrum observations and may lead to possible communication failures because users cannot foresee future channel status. Another main problem with reactive approach is the ping-pong effect which means frequent switching between the new and old channels. It happens when the measured handoff decision parameter is close to the compared threshold value. More

parameters may be employed to make more intelligent decisions to prevent this effect. In [51, 52], RSS threshold comparison in conjunction with hysteresis parameter are used to decrease the number of unnecessary handoffs in ping-pong effect. In [53], Lee et al. propose a handoff technique which considers residual bandwidth of a WLAN in addition to RSS as the criterion for handoff decisions. However, their residual bandwidth estimation technique is specific to IEEE 802.11e and can not be applied to other standards. In [54], Mohanty and Akyildiz propose a cross-layer handoff management protocol called CHMP, which is based on RSS threshold comparison for handoff decision and uses the speed of the mobile terminal to predict the delay for possible handoffs.

There exists some studies for evaluating the effect of cost parameter for handoff decision. In [55], McNair *et al.*. proposes an approach for defining the cost of handoff as a function of the available bandwidth. In [56], RSS and bandwidth parameters are used for cost estimation. In [57], an adaptive cost-based RSS approach is proposed for handoff decision in heterogeneous wireless networks. The common problem with these studies is that there exists no standard way of estimating the handoff cost in large and heterogeneous networks, therefore cost estimation in handoff decision is not practical without a standardized protocol.

Some other studies propose techniques for adding more intelligence to the handoff decision process [49, 58, 16]. In [49] an ON-OFF based channel model is defined to describe the usage status of a channel. According to this model, ON state indicates that the channel is being used by a primary user and OFF state indicates that the channel is idle. Using this model, they propose an algorithm for determining sensing mode and sensing period of a CR to minimize the delay in finding an available channel. They also propose an ordering methodology for the detected candidate channels to minimize reconnection delay. In this work, their main focus is achieving faster channel discovery. In [58], the authors use the ON-OFF model and propose a technique for maximizing throughput in TV-broadcasting networks. The work in [16] proposes an algorithm for minimizing disturbance to primary users and providing fast recovery by predicting future spectrum availability in advance and switching channel before a primary user is detected.

Previous studies on handoff mostly focus on determining the handoff time, increasing bandwidth usage, achieving faster channel discovery, and minimizing disturbance to primary users. Our focus in this thesis is different from previous studies in that we focus on minimizing the number of handoffs. In addition, none of the previous studies consider user behavior as a decision parameter for handoff, however we use user connection history to predict the possible length of the next connection and select an available channel according to that prediction. We propose two channel selection algorithms to select the channel where CR user can continue communicating for the longest period of time before a new handoff is required. We use a channel model similar to ON-OFF model proposed in [49], however we also consider the existence of other secondary users in the channels. Figure 2.2 summarizes our approach briefly.

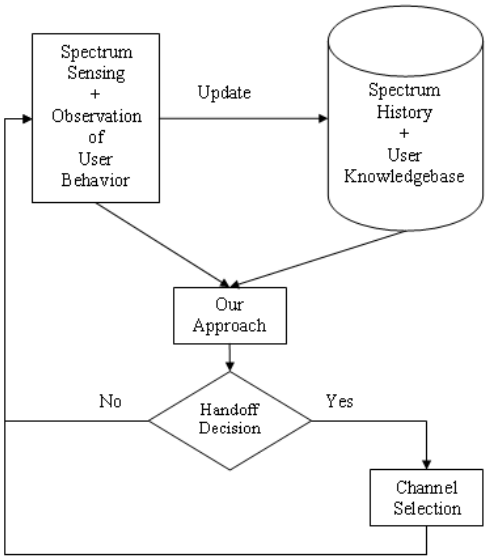

Figure 2.2. Our Approach

As shown in the figure, our approach is based on two main arguments. The first one is spectrum history and the second one is user knowledgebase. Spectrum history keeps the allocation pattern of each channel and it can be updated by using one or more of the previously proposed ways such as cooperative or non-cooperative sensing. It is up to the sensing power of the mobile terminal and the requirements of the application. The second argument is the user knowledgebase which is neglected by previous studies. It is the database of the user connection history and it is used to predict the length of the next connection request of the user. User knowledgebase and spectrum history are

critical for the success of our channel selection algorithms. They are used to predict near future spectrum availability and allows the CR to intelligently adjust its spectrum usage according to the user behavior in order to prevent interference with the primary users.

Another main difference of our thesis from previous works is that most of the previous studies provide only software simulation or hardware testbed results and neglect the software design and implementation details of their approach on a real SDR. However, we consider software design challenges and provide an infrastructure based SDR implementation. Our approach is based on SCA standard which is explained and compared with alternatives in the next part. Our design consists of several software layers where the infrastructure based layered approach increases software quality in terms of portability, scalability, reusability, and so on. We also evaluate our implementation from several aspects in the evaluations chapter.

2.1.2. Software Architectures for SDR

Significant effort is being invested in developing software architectures and interfaces to standardize SDR. Today we do not have the ideal CR, however we have some sort of SDRs. It is a fact that a mature SDR architecture plays a dominant role for CR, because CR is also a software radio and it is the enabler technology of the CR. This section summarizes existing SDR software architectures and presents a comparison of them in terms of pros and cons and discusses their applicability to CR.

The term architecture usually refers to the design of a system. When considering digital architectures such as radios it may refer to the design of hardware or software parts of it. In this thesis we are going to focus on software architectures.

The field of computer science and especially software systems has come across problems associated with complexity since its formation [59]. The basic principles to handle the complexity problems usually are based on the techniques of reducing complexity through abstraction and separation of concerns. The idea usually depends

on the algorithm of "divide and conquer" [60]. It means sub-problems are easier than the main problem itself. Dividing a big software into software components, determining externally visible properties of those components and finally deciding the relationships between them may be considered as an application of divide and conquer methodology to software architectures.

Although the separation of concern techniques are useful when considering software architectures they are not enough to solve all problems. As a maturing discipline with no clear rules to build a system, designing software architecture is still a mix of art and engineering. Determining the best practices, applying sets of design patterns and styles are inevitable to get a matured final design. Description languages, and formal logic are also considered to be other requirements.

Every system is unique due to the nature of the requirements it supports, as such the degree of quality attributes exhibited by a system such as fault-tolerance, extensibility, backward compatibility, reliability, availability, security, maintainability, usability, and such issues will vary with each implementation [61]. Effective software design requires considering issues that may not become visible until later in the implementation.

In the following subsections some of the common architectures that have been proposed to be applied to the SDRs and some design techniques are discussed in more detail.

2.1.2.1. Software Communications Architecture (SCA). Ideal SDRs are the most flexible communication devices that can operate as different radios by installing new applications. Since the hardware and software components of this evolving concept have not been fully standardized, the implemented SDR applications have unanswered question marks on portability, reconfigurability, reusability and such issues. Software Communications Architecture (SCA) [62] is a standard that is intended to address these problems.

While the SCA was originally intended solely for military purposes such as solving interoperability problems between different units of the army (aircraft to aircraft, aircraft to ground, command center communications, handheld, mobile terminals, navy, underground, underwater, space and so on) it is slowly gaining commercial viability due to the possible application areas in civil and commercial world.

The civil world has lots of similar challenges such as public safety problems (Interoperability problems between different units of the government and among the wireless equipments of various public safety organizations, operating in uncertain conditions and difficult environments both at physical and spectrum perspective and so on).

On the other side, the commercial communication technology companies are also seeking ways to solve reconfigurability, reusability and portability problems. A brand-new product becomes obsolete so fast because of the rapid changing rate of new technologies and standards. The expectations in commercial world is growing with the race of putting all the capabilities(GSM, 3G, GPRS, EDGE, WiMax, Wi-Fi, mobile TV, GPS, mobile applications, multimedia applications and so on) into a single device. Hence, the communication hardware and software become more and more complex by the time. As a result, the need for the existence of common standards that will satisfy the demands of such complex devices gains more importance.

The Joint Tactical Radio System (JTRS) [63] of United States Department of Defense Joint Program Office (JPO) has initiated a series of programs to generate a common specification for SDR platforms. The initiative started in mid-1990 and evolved into the Software Communications Architecture. Although, there have been prior radio infrastructures and architectures, the SCA is the first such specification that represents the contributions of many of the key radio system manufacturers and it is considered to be the first most complete, and well-defined architecture available for SDRs so far. The SCA standard is also supported by non-profit civil organizations from all over the world such as SDR Forum [64] and Object Management Group (OMG) [65].

It basically provides high level CORBA [66] interfaces for SDR applications. These interfaces contain common operations that every SDR application should implement. It also provides some guidelines to define behavioral details of these operations. Although the SCA standard tells developers "What to implement ?", it does not deal with "How to implement ?". Most of the implementation details are intentionally left to the developers in order not to make the SCA harder to understand.

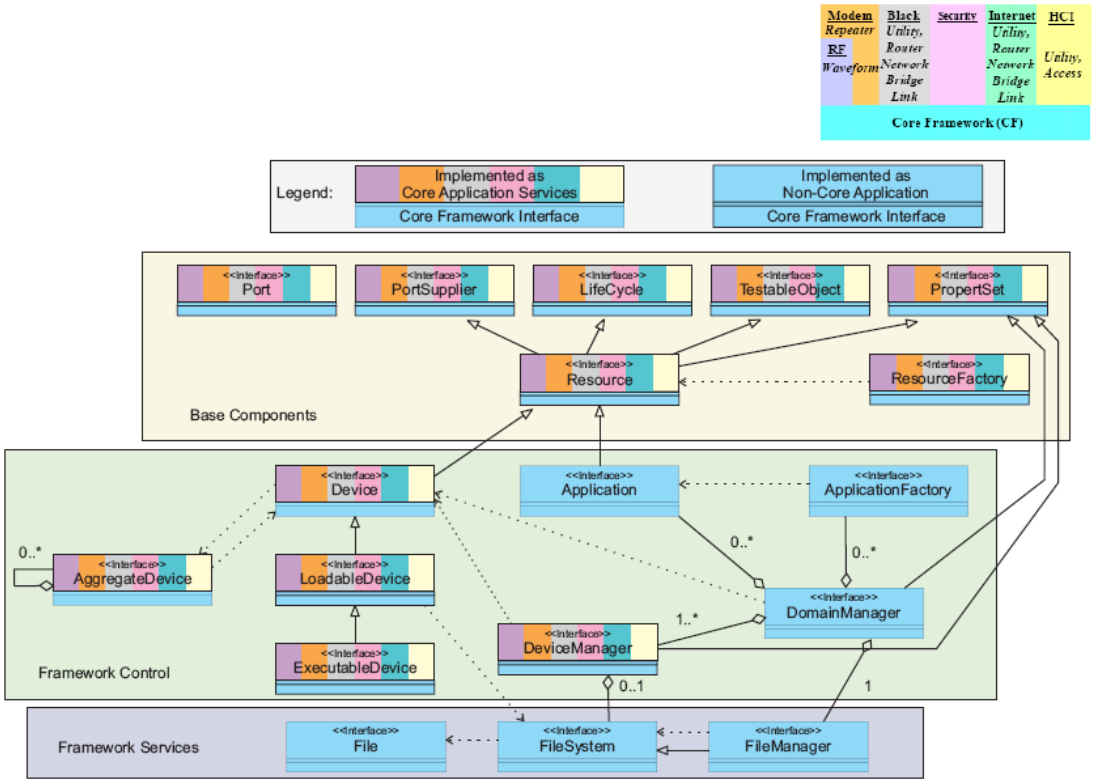

Figure 2.3. SCA Interfaces

As shown in figure 2.3, SCA standard defines three types of interfaces as defined below:

- **Base Application Interfaces:** The Base Application Interfaces are implemented by each SCA compliant component that can be managed by the radio. They represent the basic building blocks of the waveforms and implemented by each waveform component. The interfaces in this group are:
  - **Port:** Application components are connected to each other by generic Port interface. They can extend it to add more functionality. This interface

provides plug and play behavior to SCA applications. The application components that implement Port interface can be connected or disconnected to each other. Port interface provides scalability to the applications. By connecting different components to each other it is possible to create new applications.

  - **LifeCycle:** The components that implement this interface can be initialized or released after instantiation.
  - **TestableObject:** This interface provides a way of black box testing for the components.
  - **PropertySet:** This interface adds the capability to configure and query the properties of the components.
  - **PortSupplier:** This interface is used to obtain a specific port on a component.
  - **ResourceFactory:** This interface defines a factory to create new instances of the same component that implements Resource interface.
  - **Resource:** This interface groups the PortSupplier, LifeCycle, TestableObject and PropertySet in a single interface and also adds capability to start and stop the component.

- **Framework Control Interfaces:** These interfaces provide the control of the radio system. The waveform applications can reach the operating system, devices and other layers through these interfaces. The interfaces in this group are:
  - **Application:** SDR applications (Waveforms) consist of several components which implements Resource interface. These components are managed by the Application interface. This interface is used to group the application components and manage the life cycle of the waveforms. a SDR may have several applications installed at the same time.
  - **ApplicationFactory:** This interface provides a factory to build new applications. It is responsible to read configuration file (Software Assembly Descriptor) of the waveform application and create the application as defined over there.
  - **DomainManager:** The DomainManager is responsible from the control and management of all other interfaces and the radio system. It provides

methods to install and uninstall new applications. It also keeps the list of devices, applications and services that the radio includes. DomainManager is the first starting component when the radio is booting and the last component when the radio is shutting down. All other components locate the DomainManager and register themselves to it when they start running and unregister themselves when they are releasing.

  - **Device:** This interface represents the hardware component that has some capacity and can serve to waveform component. For example a sound card can be regarded as a device. It is responsible from initialization, termination of the hardware it represents.
  - **LoadableDevice:** This interface adds loading and unloading capabilities to the Device interface. Any device that can load files has to implement LoadableDevice interface. FPGA can be an example to this interface.
  - **ExecutableDevice:** This interface extends LoadableDevice interface and adds execute and terminate operations. If a device can load and run some code that it has to implement ExecutableDevice interface. GPP or DSP are the examples of the ExecutableDevice.
  - **AggregateDevice:** This interface groups other Device interfaces and is responsible to manage them. Some electronic cards that include Device, LoadableDevice and ExecutableDevice or any combination of them at the same time can be regarded as the AggregateDevice. For example an expansion card that includes two FPGA and one DSP can be given as an example of AggregateDevice.
  - **DeviceManager:** This interface is responsible to manage devices and services (Such as log service) of the radio. It provides methods for registering or unregistering of devices and services. There may be several DeviceManagers at the same time in a radio. The general methodology is to implement one DeviceManager for each node that has an IP. For example, a SDR that has two processing units that are connected to each other over network should have two DeviceManagers, one each side.

- **Framework Services Interface:** The Framework Services Interfaces provide the system services. They are:

  - **File:** This interface represents a file on the radio. It provides file manipulation operations such as read and write for any kind of file. This interface provides portability to the waveforms by providing them to access the files through a common interface. For example a waveform that wants to write a file does not have to worry about the operating system as long as it reaches files through this interface, because each platform implements File operations according to their platform specifications so that waveforms become more portable.
  - **FileSystem:** This interface represents makes the underlying physical file system transparent to the applications. Different file systems like FAT32, NTFS, and Unix File System can be used with the same interface and the user does not even know where the physical file actually resides in the system because of location transparency of CORBA.
  - **FileManager:** The FileManager provides another level of abstraction so that multiple, even distributed FileSystems may be accessed and managed through a FileManager.

Figure 2.4 shows the hierarchy between these interfaces [62]. As shown in the figure DomainManager is at the top level and manages all other staff. DeviceManager is responsible from Devices and Services. It may include several of them. FileManager includes FileSystems which manages Files. On the other side, the Applications of the radio are managed by ApplicationFactory and each Application is composed of the combinations of several Resources or ResourceFactories.

Figure 2.5 shows the layers of an SCA platform. The software layers in the SCA can be listed as follows:

- **Hardware:** Hardware includes the physical set of devices and components that comprise the radio set such as GPP, DSP, FPGA, NIC, ADC, DAC, DDC, DUC, RF components and so on. They are usually grouped together by using a chassis.
- **Operating System:** Operating System is usually a COTS software that is responsible for the management and coordination of activities and the sharing of

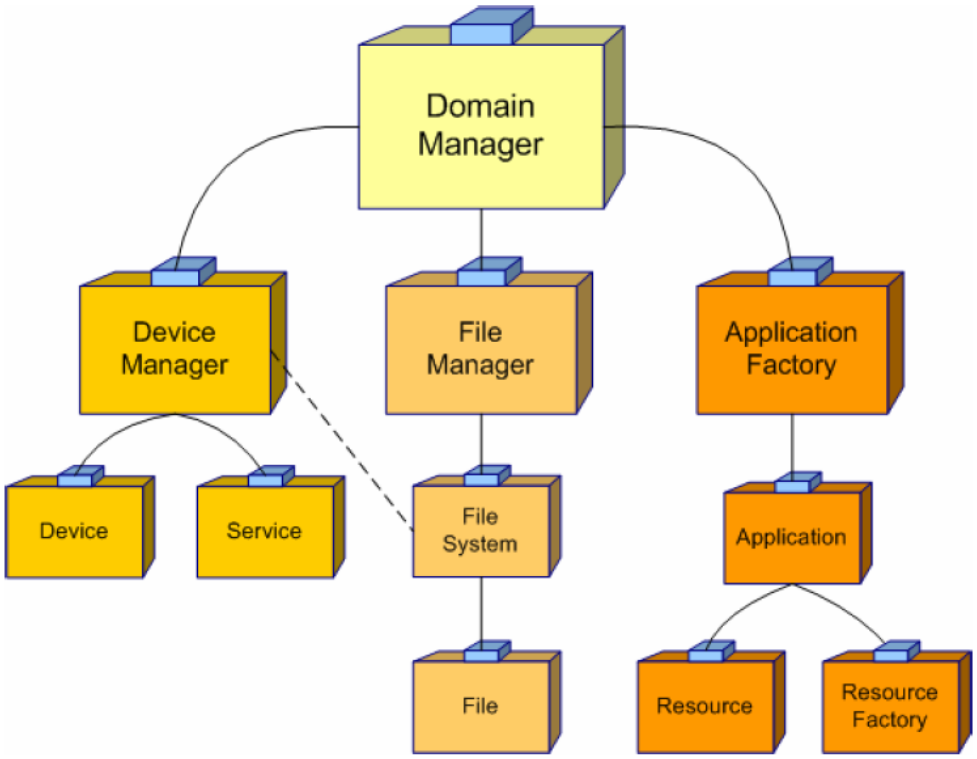

Figure 2.4. SCA Hierarchy

the resources of the computer. The operating system acts as a host for applications that are running on the machine. SCA requires operating systems to be POSIX compliant which is a standard to assist portability of applications and ensures operating systems to support a predefined set of API. VxWorks, Integrity and Linux can be given as example that complies with POSIX.

- **Hardware Abstraction Layer:** This layer includes device drivers and board support packages.
- **CORBA:** CORBA is the basic transport between the software layers of the radio set. It provides implementation independence, location transparency. A CORBA-based application written in an ORB supported language and running on an ORB supported platform can interoperate with another CORBA-based application. ACE/TAO, ORBExpress, OMNIORB, eORB are some examples of ORB middleware.
- **Core Framework:** Core framework is a standardized run-time layer to manage waveforms. It is the implementation of SCA interfaces in order to assist deploying, configuring, controlling, and monitoring the hardware and software applications

within an SCA-based radio system. It hides the technical details of the hardware, operating system and device drivers to the waveforms and makes them more platform independent.

- **Waveform:** Waveforms are radio applications that manage the operations of radio from antenna to the data input. They are composed of binary files and xml files. The binary files are the actual implementations of the operations. The XML files include the configurations of the binaries such as CORBA connections, dependencies, initial configuration values and so on.

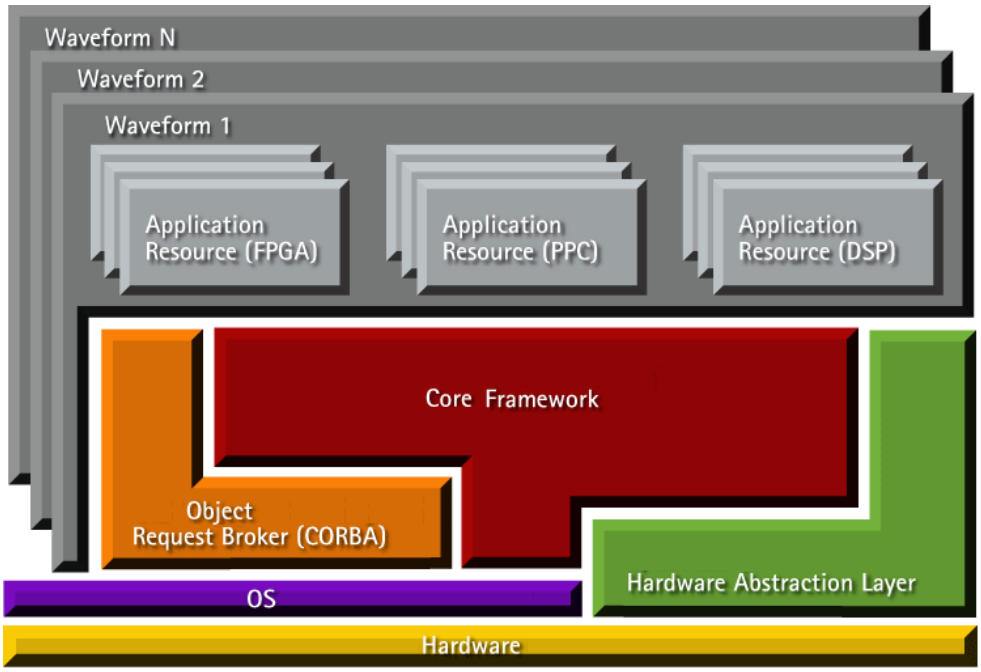

Figure 2.5. SCA Layers

The main goals of the SCA can be listed as follows [67]:

- **Common Open Architecture:** It is an open standard which is also based on open standards such as CORBA, XML, POSIX. Using common open architectures promotes competition, interoperability, technology, insertion, quick upgrades, software reuse, and scalability.
- **Multiple Domains:** It is targeted to support a wide variety of domains such as airborne, fixed, maritime, vehicular, and handheld.
- **Multiple Bands:** SCA based radios aims to interoperate or replace different

radios running at wide range of frequencies.

- **Software Reusability:** It also aims to maximize software reusability to support plug-n-play behavior within waveforms components.
- **Compatibility:** SCA is intended to generate radios to communicate with legacy radios.
- **Upgrades:** It aims to enable transparent technology insertion.
- **Networking:** SCA provides seamless integration with legacy network protocols and supports wideband networking capabilities.

2.1.2.2. OMG SW RADIO. Object Management Group (OMG) [65] is a large organization that defines world-wide accepted standards. They define software models such as Platform Specific Model (PSM) and Platform Independent Model(PIM) and some sort of architectures such as Model Driven Architecture(MDA) [68] to facilitate software portability and interoperability.

Platform Specific Model can be defined as the software that is linked to a specific technological platform such as a specific programming language or operating system. The PSM is indispensible for the actual implementation of a system in order to make it run. On the other side, Platform Independent Model can be regarded as the model of a software that is independent of the specific platform that is used to implement it. The PIM and PSM approaches are most frequently used in the context of the MDA approach which is the model driven engineering vision of OMG. The main idea is that it should be possible to use a Model Transformation Language (MTL) to transform a PIM into a PSM. In order to achieve this transformation, one can use a language compliant to the newly defined QVT standard.

SWRADIO Domain Special Interest Group (DSIG) [69], under OMG, is currently under development of a PIM for the SDRs. Their standard is based on MDA approach and they try automatic conversion of the PIM to PSM.

Basically, their PIM model consists of UML [70] diagrams for SDR components.

Their model investigates the SDR by dividing it to the following sections:

- **Software radio:** The UML Profile for software radio provides the required language to support modeling waveform applications, platform components, and communication channels independent of specific technology choice. It brings the flexibility to perform specific transformations of these elements in the future directly into the implementation technologies required. For example, waveform application components might initially be implemented in C++ for the Linux operating system running on a Pentium general purpose processor (GPP). Later, possibly due to technology upgrades the same waveform components might be needed in VHDL to support a FPGA or in C to support a DSP. Since the specification does not force technology choice, the developer can transform the architectural concepts found in the specification to support the VHDL (FPGA) or C (DSP) programming language.
- **Data link:** The PIM model in this layer defines the link layer control (LLC) and media access control (MAC) for communication needs.
- **Physical layer facilities:** The model in this layer defines the functionality to convert the digitized signal into an RF wave, and conversely, to convert an RF wave into a digitized signal for processing. Also, the PIM includes facilities for frequency tuning, filters, interference cancellation, analog/digital conversion, up/down conversion, gain control, synthesizer etc.
- **POSIX profiles:** The OMG SWRADIO specification defines a set of mandatory functions called Application Environment Profile (AEP) [71] based on the Portable Operating System Interface (POSIX) [72] definition. In addition to this, a lightweight Application Environment Profile (LwAEP) is also defined. The LwAEP is constrained version of the AEP and is targeted at environments with limited resources such as embedded processors like DSPs, FPGAs and microcontrollers.

The PSM for the PIM specified in the OMG SWRADIO specification at this time for the Component Framework Profile are CORBA and XML. The UML interfaces that are defined in PIM are transformed into CORBA interfaces and the component

descriptors are transformed into similar XML descriptors such as DTD.

2.1.2.3. OBSAI. Open Base Station Architecture Initiative (OBSAI) [73] is founded by base station vendors Hyundai, LGE, Nokia, Samsung and ZTE in September 2002. After its foundation many other commercial wireless industry companies has joined and supported this initiative.

The idea behind OBSAI is to create an open market for cellular base stations to reduce development effort and costs associated with creating new base station products and producing devices that can be interchanges between the companies supporting OBSAI.

OBSAI defines the Base Transceiver Station (BTS) concept and standardizes the architecture, function descriptions and minimum requirements for integration of a set of common modules into a BTS. A Base Transceiver Station (BTS) has four main blocks or logical entities:

- **Radio module:** It includes RF transceivers, amplification and conversion between digital baseband and analog RF signals. The radio module receives signals from portable devices (via the air interface) and converts them to digital data.
- **Processing module:** It includes channel modems and the baseband processing for the air interface. This module processes the encoded signal and brings it back to baseband.
- **Control module:** This module is used to coordinate the three other modules.
- **Transport module:** It provides adaptation between external network and internal interfaces. It is responsible for relaying the date to the terrestrial network.

The advantages of OBSAI can be listed as follows:

- It defines an open, standardized internal modular structure of wireless base stations. This allows next-generation radio base stations to be built using shared

platforms and modules, available on an open market.

- It defines a set of standard BTS modules with specified form, fit and function. This allows manufacturers to focus their research and development efforts on their core competencies and to buy selected radio base station modules from each other and from other module vendors.
- It defines open, standards-based internal digital interfaces between BTS modules to assure interoperability and compatibility. This approach to writing the set of compatibility specifications is intended to provide the BTS integrator with sufficient flexibility to respond to differences in access technologies, configurations, reliability, capacity, etc.
- It supports different access technologies such as GSM/EDGE, CDMA2000, WCDMA or IEEE 802.16/WIMAX that are currently on the market.

2.1.2.4. CPRI. Common Protocol Radio Interface (CPRI) [74] is an other initiative which has been founded by Ericsson, Huawei Technologies, NEC Corporation, Nortel Networks and Siemens in response to OBSAI.

CPRI has a much narrower focus compared to OBSAI. Unlike OBSAI, CPRI does not specify mechanical or electrical interfaces and it only focuses on the link between the radio frequency (RF) and channel cards found in the base station. CPRI divides the radio base station into a radio and a control part and it specifies one new interface.

The key goals of CPRI can be listed as follows:

- It provides an openly available specification.
- It aims to achieve shorter time to market.
- It enables base station manufacturers and component vendors to focus their research and development efforts.
- It allows for new architectures and is not limited by module dimensions or a pre-defined function split.

2.1.2.5. Comparison of Software Architectures. All of the specifications try to facilitate interoperability, technology insertion, quick upgrade capability, software reusability and scalability. However these standards cannot achieve them at the same level. When the efforts to standardize SDRs are compared the following conclusions can be drawn:

- OBSAI and CPRI are mostly focused on standardization of 3G base stations, whereas the SCA and OMG SWRADIO try to support wider variety of domains.
- SWRADIO is only supported by OMG; OBSAI and CPRI are supported by commercial companies, however SCA is supported by relatively larger variety of manufacturers including military side, commercial vendors and universities.
- The SCA is defined at PSM level using CORBA interfaces and XML DTDs. In contrast, the OMG PIM and PSM for software radio components define UML profiles for modeling SDR concepts and PIM facilities that can be transformed into any technology.
- The core framework and PortTypes CORBA module interface definition language (IDL) in the OMG specification is broken into multiple files instead of condensed one monolithic file as in the SCA. This allows implementations of these interfaces to remain smaller in memory size.
- The OMG specification has started as a project which is based upon lessons learned from several SCA core framework implementations. It provides several optimizations in deployment, component connections and teardown to support lighterweight component definitions for other software radio domains. For example, OMG SWRADIO mandates LwAEP which is lighter version of AEP that is recommended by SCA.
- All of these standards are still evolving.
- Only SCA has reference implementations on the market but still these are not enough.
- None of these standards focuses directly on CR however SWRADIO and SCA can used as a base for CR.
- None of them include security aspects such as crypto sub system. However there are some working groups trying to develop security standardizations.

In summary, it can be concluded that OBSAI and CPRI are far from providing a complete standardization for SDRs. OMG SWRADIO sounds good and claims to optimize SCA, however it has no real implementation in practical and cannot go beyond of being a standard. On the other side, although the SCA has some bottlenecks such as being mostly GPP centric it is relatively more applicable and has several implementations.

2.1.3. Existing SDR Projects

There are several ongoing SDR projects from both professional and amateur world including commercial, defense, civil government organizations, regulatory agencies, and academia. They aim to accelerate the design and development activities of SDR architectures. In this section we summarize them briefly.

2.1.3.1. SCARI-Open. SCARI-Open (SCA Reference Implementation) [75] is an open source implementation of Software Communications Architecture (SCA) standard. In year 2001, SDR Forum has contracted with Canadian Research Center (CRC) to develop a framework application for SDR platforms. This project is also supported by DoD of Canada and USA. It can be considered as a vital project because of being the first well-organized SDR framework.

It is implemented in Java programming language so that it mainly targets PC platforms. It does not support embedded devices and real-time operating systems. SCARI-Open source code can be downloaded from [76]. It provides a standardized way of managing SDR applications (waveforms). Waveform is the actual communication software which includes all the algorithms and radio operations. The main tasks of SCARI can be listed as follows:

- Installing waveforms.
- Deploying waveform components to target processors.
- Configuring waveform components.

- Releasing waveform components.
- Uninstalling waveforms.

It also provides the following abstraction mechanisms for the underlying operating environment.

- **Executable Device:** It provides a standard way of executing or terminating executable waveform files and loading or unloading shared libraries, kernel modules, and drivers needed by waveform components at runtime. Loading kernel module or driver is only supported in commercial versions of SCARI core framework.
- **Audio Device:** This device provides a managed way of capturing and playing audio from the underlying hardware.
- **RF Device:** It is a control mechanism for RF hardware of CRC. It is connected to PC over serial port and enables waveforms to transmit or receive RF packets.
- **File Manager:** It provides an abstraction mechanism for the underlying operating system for the waveform components to manage file operations.
- **Log System:** It is a standardized way of logging debug, info, and error messages.

In our thesis, we use SCARI-Open as our core framework layer. We implement a waveform for SCARI-Open core framework using C++ programming language. Since SCARI-Open is implemented in Java, our waveform is a successful demonstration of interoperability between SDR layers implemented in different programming languages. Our waveform also demonstrates other CR metrics such as reconfigurability, portability, reusability, scalability, etc...

2.1.3.2. OSSIE. OSSIE (Open Source SCA Implementation - Embedded) is an open source SDR development project by Virginia Tech. It primarily aims to support research and education activities in SDR and wireless communications. OSSIE team is composed of several graduate and undergraduate students leaded by Dr. Jeffrey H. Reed and Dr. Carl B. Dietrich.

OSSIE team develops open source SDR software that includes rapid development and debugging tools, signal processing libraries, as well as SCA based infrastructure software which is written in C++ using omniORB for CORBA. OSSIE framework works on Linux, however there exist some efforts to port it onto other operating systems such as BSD, OSX, Windows and Integrity. Current version of OSSIE is 0.7.0 and can be downloaded from [77]. OSSIE runs on x86 based PCs and uses USRP board as the RF front end.

OSSIE implementation is not stable and complete enough, when compared to SCARI-Open project. It has many bugs and it does not implement all aspects of SCA. It is mostly because of being an ongoing project. On the other hand, OSSIE is a C++ implementation, whereas SCARI is Java. Therefore, OSSIE works relatively faster and it can be preferred as a framework in time-critical SDR projects. Apart from SCARI-Open and OSSIE there are several core framework implementations in the market such as Harris, SCARI++, ORCA, and so on, however they are commercial products and they are not open source.

2.1.3.3. GNU Radio and USRP. GNU Radio [78] is a free software development toolkit for SDR systems. It has been initiated in 1998 by John Gilmore and has been widely used by hobbyist, academic and commercial environments to implement amateur or real-world radio systems such as HDTV decoders, GPS receivers, GSM stations, FM transmitters, garage door openers, and even some studies [79] has shown that GNU Radio with the necessary RF unit can be used to attack people who has cardiac problems.

GNU Radio simulates receiver and transmitter chains that are described as blocks, where each block can be considered as a signal processing black box. It has been designed to be used on a Linux installed PC. GNU Radio is implemented with Python language, however C++ is used for performance-critical signal processing. GNU Radio supports development of signal processing algorithms using pre-recorded or generated data to avoid the need for actual RF hardware.

GNU Radio is a software project and without the necessary RF hardware it cannot be used as a real radio. Matt Ettus and his team has developed a cheap and easy to use RF transmitter called USRP (Universal Software Radio Peripheral) [80] that can be connected to GNU Radio installed PC over USB port. The USRP team has also implemented some libraries for GNU Radio to transmit data packets over USRP. It is widely accepted by the GNU Radio users as the necessary RF hardware.

The scope of GNU Radio project is completely different than the SCARI and OSSIE projects. They are intended to provide a software framework for SDR platforms, whereas GNU Radio focuses on immediate SDR functionality. GNU Radio does not include a framework and the implemented codes are completely platform dependent. GNU Radio does not offer a standard way of installing, configuring, executing, and managing SDR applications. In addition, it cannot be used in areas such as military applications where real-time communication is critical.

2.1.3.4. HPSDR. HPSDR (High performance software defined radio) [81] is an alternative to USRP and GNURadio. The project is started in year 2005 and it is still being developed by a group of radio enthusiasts. The main idea is to provide a set of low-cost SDR hardware modules that have the same interface and can be plugged in a common motherboard called Atlas. Each module is designed by different groups. Experimented users chooses among these modules and assembles them to create their own variant of radio.

HPSDR project aims to be a software and hardware project at the same time. However, software parts of the projects is too weak so far. In their web site, they mention that there is still much to be done in bringing HPSDR to fruition.

HPSDR offers more powerful and flexible SDR hardware compared to USRP board. However USRP is cheaper and includes some libraries for GNU Radio software, whereas HPSDR requires more expertise and software support is weak. In addition, GNU Radio and HPSDR are intended for agile SDR development and does not provide

any kind of software architecture but some library source.

2.1.3.5. Power SDR and FlexRadio. Power SDR [82] is an open source project including DSP and hardware control functions. It has been developed by FlexRadio systems for their transceivers such as FLEX 5000-A. FlexRadio is a SDR with an interface to the host PC. It connects to PC over FireWire port which is faster than USB connection. PC part of Power SDR software is implemented in C# and it is easy to learn and modify. Although Power SDR is an open source project, FlexRadio hardware is very expensive compared to its alternatives such as USRP. Power SDR software has been also adopted by HPSDR team to support their hardware.

2.1.3.6. Other SDR Projects. There are many SDR projects in the market in addition to the projects above, however most of them are proprietary or military solutions. Adapt4, LLC, Aeronix, Aerospace Corporation, Agilent Technologies, Air Force Research Laboratory, Anritsu Corporation, Aselsan, A.S., ASRC Aerospace Corporation, Astrium, Ltd., AT&T, AT&T Labs, BAE Systems, Battery Ventures, Bharat Electronics Limited, Boeing, Booz Allen Hamilton, C-DAC - Centre for Development of Advanced Computing, Cinterion Wireless Modules, Cognitive Radio Technologies, LLC, Communications Research Centre Canada, Datasoft Corporation, Datron World Communications Inc., DEAL-DRO (Defence Electronics Applications Laboratory), Diversified Technology, Inc., DRS Signal Solution, DSO National Laboratories, EF Johnson, EID, Elbit Systems Land and C4I Tadiran, Elektrobit, ENSTA, Etherstack, ETRI Korea, FMV, Swedish Defence Materiel Organization, France Telecom, Fraunhofer, GE Fanuc, General Dynamics C4 Systems, Hanyang University, Harris Corporation, Hitachi Kokusai, Hitachi Ltd., Hypres, IDA (Institute for Defense Analyses), IMEC, Indra Sistemas, Infineon Technologies, Innovative Concepts, Institute for Infocomm Research, ISR Technologies, ITT Communications Systems, L-3 Communications, Lyrtech, Mathworks, The Mercury Computer, MIT - Massachusetts Institute of Technology, MITRE, Motorola, NASA Glenn Research Center, National Institute of Information and Communications Technology, National Public Safety Telecommunications Council (NPSTC), Navsys Corporation, NEC, Oak Ridge National Lab., Objec-

tive Interface Systems, Omniphase Research, Pentek, PrismTech, QinetiQ, QuickFlex, RadioFrame Networks, Raytheon Systems, Reservoir Labs, Rockwell Collins, Rohde & Schwarz, Royal Institute of Technology (KTH), Sandbridge Technologies, SCA Technica, Selex Communications, Shared Spectrum Company, Skybridge Spectrum Foundation, Southwest Research Institute, Space Coast Communications, SPAWAR Systems Center, Spectrum Signal Processing, ST Microelectronics, Stevens Institute of Technology, Synopsys, TDK Corporation, Telefunken Racoms, Thales, TNO, TRDA (Taiyo Yuden R&D Center of America), Tubitak, Tyco Electronics, Ultra Electronics-TCS, Universitaet Karlsruhe, Universitat Politcnica de Catalunya, University of California San Diego, University of Oulu, Vanu, Inc., ViaSat, Viettel Technologies, VIP Mobile, Virginia Tech, VIStology, Inc., Xenotran Corporation, Xilinx, Yokohama National University, Zeligsoft are among the organizations that are involved in SDR projects [83].

2.2. Overview of Cognitive Radio

In this section, we overview CR in more details. The reasons to achieve cognition are summarized while the general working principles, the lifecycle and the logical components to create the backbone of a CR are explained. After listing that the challenges that have to be met to build a CR, we explore the required hardware components briefly. We discuss the architecture of a software design that achieves the goals of CR and supports the required hardware.

2.2.1. Cognitive Radio

The topic of CR has gathered great deal of attention in the past several years. Opinions about the level of sophistication necessary to qualify a system as cognitive is open to discussion and has no exact answer at least for the moment. The sharp increase in the number of wireless devices and different technologies forces the radios to evolve from completely hardware based digital equipments to software defined and highly intelligent communication systems.

Figure 2.6 shows the evolution of software radios [84]. Software radios have several

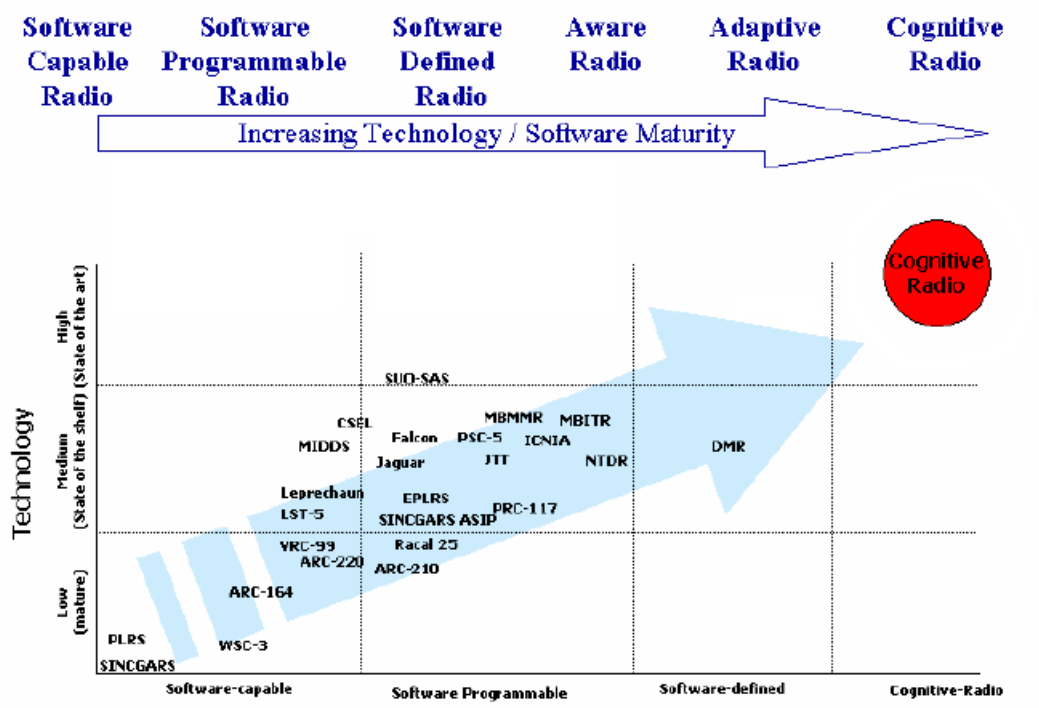

Figure 2.6. Evolution of Software Radios

levels of maturity definitions ranging from software capable radio to CR. In fact, these definitions do not have sharp borders. However, it can be claimed that the level of complexity, intelligence, and flexibility increases as we move towards CR.

In software capable radio, at least one radio component can be reconfigured by changing its software. Software programmable has more software processing capability. In SDR, radio behavior is completely defined by the software. In addition, an aware radio is assumed to have sensors and it is aware of the environment (or at least subset of the environment). An adaptive radio is assumed to be aware of its environment and it is capable of changing its behavior in response. In this classification the CR includes additional features beyond adaptation such as automatically adjusting its behavior or operations to achieve desired objectives by checking much more parameters. As of today, we have no fully CR; Mitola anticipates that we will not have it until 2030. However, we have some sort of software radios, aware radios, or adaptive radios. SDR is the basic building block for a CR. Each CR can be regarded as a SDR but the reverse is not always true.

CR proposed by Mitola in 2000, is a newly emerging technology to make use of radio frequencies in an opportunistic driven basis [7]. It is defined as "adaptive, multi-dimensionally aware, autonomous radio (system) that learns from its experiences to reason, plan, and decide future actions to meet user needs" by the SDR Forum Cognitive Radio Working Group [85]. In fact, there are several definitions in literature for CR. These definitions depend on the set of parameters taken into account in deciding on transmission and reception changes.

2.2.1.1. Cognitive Radio Benefits. The CR concept determines the current limits of our expectations for a wireless communication device. The development of CR devices will take time, but the effects on wireless communications is expected to be significant. The benefits of CR can be investigated from both user and service provider perspectives. The benefits from user perspective can be listed as follows:

- CR promises to improve spectrum efficiency by detecting the holes in the spectrum, switching communications to these holes, and then moving away as soon as a licensee begins transmitting. In other words, CR is expected to utilize any unused spectrum as a secondary user without interfering with the primary users. This approach provides a vital tool to overcome spectrum bottlenecks. As an example scenario, one might consider hot spots and disaster regions where busty traffic causes high levels of blocking and dropping rates. Although, existing GSM companies try to cope with these problems by using mobile GSM stations, they cannot handle sudden bursts. The CR offers techniques to overcome such problems by dynamically sensing and adapting to the radio environment.
- CR can understand and follow actions and choices taken by users and over time learn to become more responsive to anticipate user needs.
- CR can also negotiate with several service providers to connect the user in an optimal way in terms of QoS metrics like bandwidth, connection time, error rate and other criteria like pricing. For instance, a mobile phone user may determine his communication preferences and his cognitive capable phone can take care of determining the best communication alternative that meets user requirements.

From a user perspective, this means saving money and time without worrying about the technical details.

- CR increases interoperability between communication devices by automatically recognizing the communication standards and adapting them. Adaptation may require automatically downloading necessary software over the air. For instance, if a user travels to another country where communication standards are different, his CR is expected to adapt itself by recognizing the standards of the country visited.
- CR provides dynamic adaptation in many parameters (frequency band, bandwidth, time, power, modulation level, code, etc).
- If needed, CR may provide information about internal and external environment to make the user aware of them.
- CR works autonomously. In other words, it does not require user intervention to operate so that the life becomes simpler for users.

From service providers point of view, the following benefits can be listed:

- CR helps reconfiguring networks to meet current capacity and coverage needs by improving operational efficiency and calibrating the network.
- CR helps prioritizing network resources.
- CR avoids jamming other users while transmitting with sufficient power to overcome ambient interference.
- CR encourages commercial companies to develop applications on top of cognitive frameworks to support multi-media networking, band sharing, emergency services, broadband wireless services, etc.
- Since utilizing CR optimizes network usage, the bandwidth increases dramatically. Therefore, users can access multimedia contents, for example, in shorter waiting times which yields faster introduction of new multimedia services.
- CR helps to adapt the radio and its emissions without user intervention so that prevents human-made faults.
- CR can be useful in public safety and military cases to solve communication problems between different units.

2.2.1.2. Cognitive Radio Challenges. The basic function of the CR is to learn the users needs, evaluate the internal and external state of the radio, select the best alternative action, and perform the selected changes. In this regard, the CR is expected to perform the main functions below:

- **Spectrum sensing:** The spectrum holes are detected and licensed users are determined.
- **Spectrum management:** The best available channel is selected by analyzing the spectrum holes against interference, path loss, and error rate.
- **Spectrum sharing:** Access to the selected channel with other users is coordinated.
- **Spectrum mobility:** Selected channel is vacated in case of licensed user detection.

These operations are considered to be the basic missions of a CR. To be able to perform these operations, the CR is expected to be aware of its environment and match requirements of a higher layer application or user with the available resources of the radio equipments. Achieving these concepts is tightly coupled with the solutions on each challenge of the CR. Potential challenges of CR can be listed as follows:

- Hardware Challenges
  - How to achieve wideband sensing capability with the RF layer.
  - How to build fast, power efficient, small form factor and cheap computing platforms.
  - How to achieve very fast and run-time reconfigurable signal processing power on embedded processors (DSP/FPGA).
  - How to build ADC/DAC that will perform faster conversion between analog and digital.
  - How to achieve more sampling rate (Nyquist rate).
  - How to achieve higher resolution (Precision).
  - How to build smart antennas that will support CR.
  - How to produce batteries with longer life time.

  - How to handle the complexity of the devices.
- Software Challenges
  - How to sense very wide spectrum in the shortest time.
  - How to collect user and environment parameters.
  - How to achieve adaptation.
  - How to achieve awareness.
  - How to develop multi-mode, multi-functioning software.
  - How to achieve radio reconfigurability.
  - How to achieve software portability.
  - How to achieve component reusability.

2.2.2. Hardware Mapping of Cognitive Radio

Understanding the hardware architecture behind a CR is crucial to be able to evaluate the software that will run on it. In this section, generic hardware components that exist in a CR architecture is discussed briefly.

A CR consists of an analog RF front-end attached to the signal processing unit. Figure 2.7 shows the main components of a CR transceiver and the analog RF front-end.

The main components of the RF front-end of a CR and their functions can be explained as follows:

- **RF filter:** As the name implies this component filters the received signal to select the desired band.
- **Low noise amplifier (LNA):** The LNA minimizes the noise and amplifies the desired signal.
- **Mixer:** The mixer mixes the received signal with locally generated RF frequency and converts to the baseband or the intermediate frequency (IF).
- **Voltage-controlled oscillator (VCO):** The VCO generates a signal at the desired frequency for a given voltage to mix with the incoming signal.

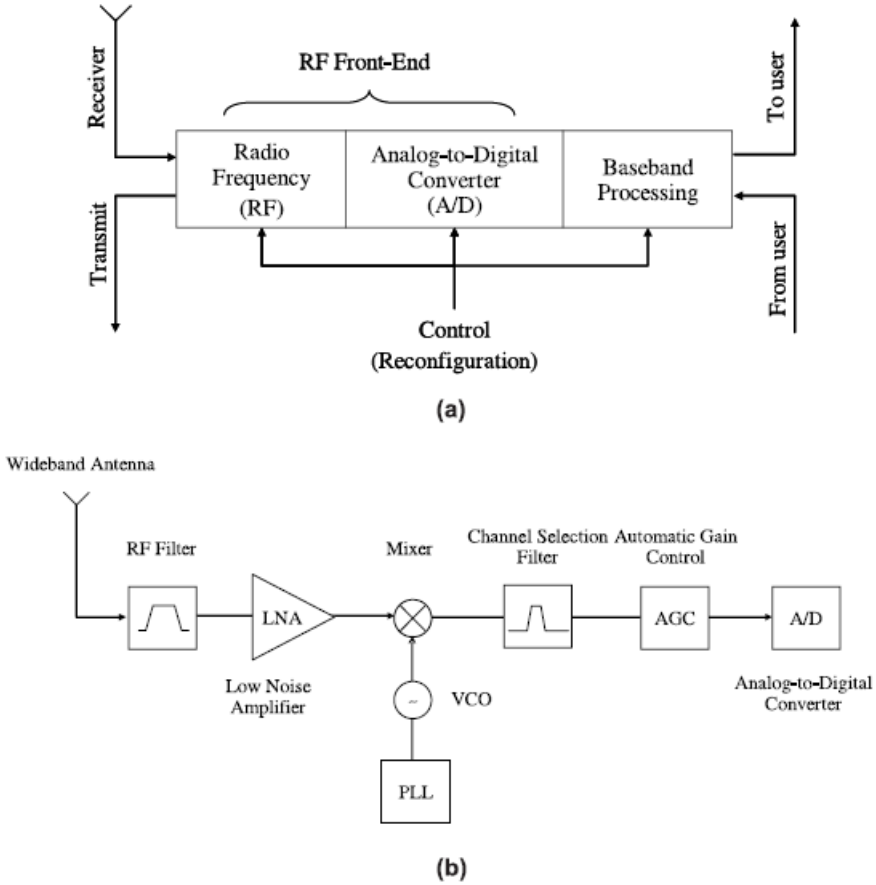

Figure 2.7. Physical architecture of the CR [86, 87]: (a) CR transceiver and (b) wideband RF/analog front-end architecture.

- **Phase locked loop (PLL):** The PLL is used to ensure that a signal is locked on a specific frequency. It can also be used to generate precise frequencies with fine resolution.
- **Channel selection filter:** It selects the desired channel and rejects the adjacent channels.
- **Automatic gain control (AGC):** The AGC is responsible for keeping the gain or output power level of an amplifier constant over a wide range of input signal levels.

In this architecture, the RF front-end receives a wideband signal and sends it to the high speed ADC. The ADC, converts the incoming signal to the digital format that can be processed by the base band processing units where the DSP or FPGA resides. Base band processing unit is assumed to be completely software defined and can be

reconfigured by loading another application.

The key challenge in building the hardware components of a CR is the successful detection of weak signals of licensed users over a wide spectrum range accurately. RF hardware for the CR should be able to tune to any part of a wide range of frequency spectrum. It is also expected to perform real-time measurements about spectral features of the radio environment. In addition, in ideal CR layout, ADCs should be directly connected to the antenna and they are expected to convert the wideband analog input to a digital format (stream of numbers), which is not feasible at least for now without using extra components in between such as preamplifiers, digital down or up converters. From this point of view, the implementation of RF wideband front-ends and ADCs are extremely critical and challenging.

2.2.3. Software Mapping of Cognitive Radio

In this section, we discuss the architectural design of the CR software. We summarize the life cycle of CR and investigate current design models in terms of pros and cons. Finally, we map the software architecture to the CR requirements.

The term "*life cycle*" describes the sequence of states that a CR passes through in a repeative manner during execution. The four basic stages of CR life cycle are summarized in Figure 2.8 [88, 89].

2.2.3.1. Sensing stage:. Sensing provides a radio device an awareness of its own internal status and the representation of the external world. The level of cognition depends on the level of awareness. The level of awareness, on the other hand, depends on the number of parameters that have been collected both from internal and external world. The list below provides some of the parameters that can be monitored by the radio environment:

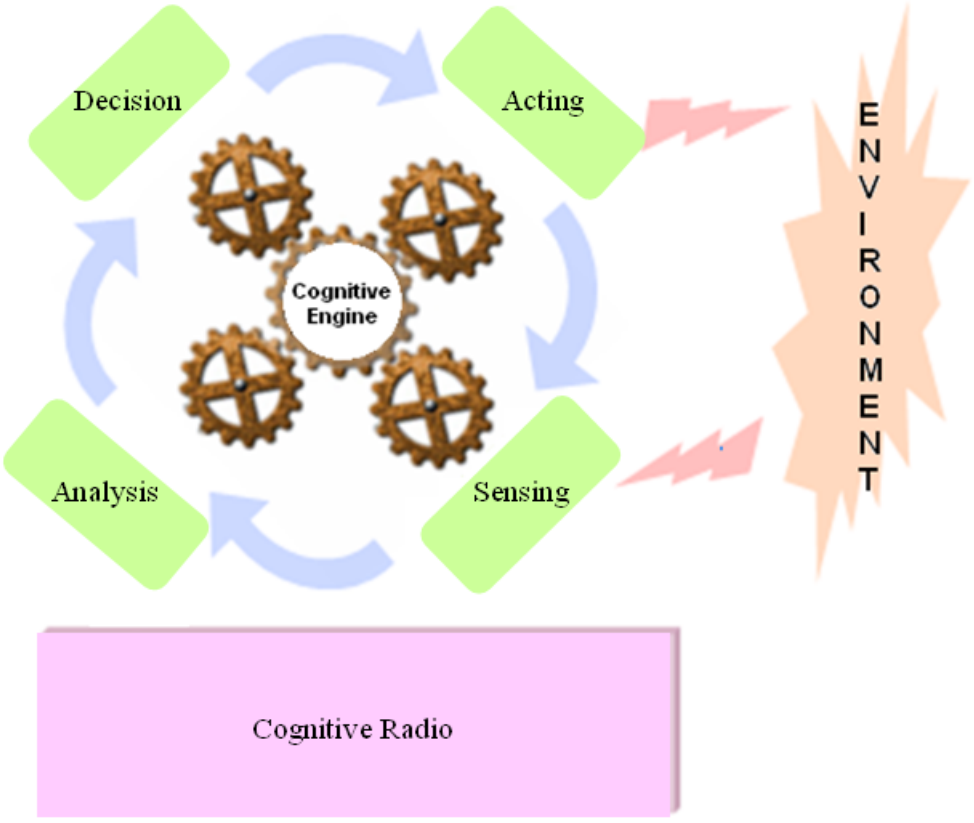

Figure 2.8. Life Cycle of CR

- Internal environment
  - Radio hardware resources
    - Processing power
    - Memory
    - Battery power.
    - RF/Antenna
    - Other available hardware and their capacities
  - Radio software resources
    - Installed waveforms and their capabilities/characteristics
    - Available software modulations
    - QoS requirements of applications and data rates
    - Local policies, regulatory, and other operating restrictions
    - Other available software
  - User preferences
    - User behavior(previous decisions and mistakes)

    - QoS metrics
    - Preferred networks
    - User licenses
    - Fiscal issues
- External environment
  - Available spectrum
  - Geolocation
  - Infrastructure
  - Channel
  - Interference
  - Types of signals around (signal characterization)

A CR device monitors these parameters and creates a database of the collected information. At this stage, the radio does not try to assess the parameters. In the sensing stage, only the parameter values are collected and the radio switches to the decision stage. In this stage, the format of the collected parameters and their storage methodology is considered to prevent capacity overflows and performance bottlenecks. Also, collected parameters can be used to create a user behavior, history which can be used while deciding future actions.

To be able to collect a variety of parameters, the radio is expected to include a variety of sensors. The technology behind the sensors such as their speed, power consumption, physical volume, and weight constitutes a crucial problem when the small form-factor of portable CR is considered. For vehicle-radios or navy-radios some of these items, such as weight may be considered to have secondary importance.

2.2.3.2. Analysis stage:. In this stage, the parameters collected in sensing stage are analyzed from a semantic point of view. The analysis process translates the set of parameters that has been collected by using various sensors to a concrete result, which describes the current state of the radio. The following can be analyzed in this stage of the cognitive life cycle:

- Detecting the holes in the electro-magnetic spectrum.
- Detecting primary users.
- Detecting interference.
- Determining the geographical location of the radio such as urban area, sea, forest, space, underwater, air.
- Analyzing QoS metrics.
- Calculating the distance of the radio to the other radios or stations.
- Determining the types of signals around by using signal characterization techniques. It may include, the modulation, power level, coding, etc.
- Determining local policies such as local formal regulations.
- Analyzing capacity of the radio such as the physical memory or the processing power that can be used to install other applications.

The list above can be extended depending on the type and the capabilities of CR. There is a trade off between deeper or faster analysis. It is the performance criteria of the application that determines the level of complexity of the analysis since most of the CR applications do not require considering all of the parameters. On the other hand, as more analysis is performed, it is clear that better results can be obtained. The output of this stage is used as an input parameter for the next stage.

2.2.3.3. Decision stage:. In the decision stage, the CR chooses the best action to be executed by evaluating the results of the analysis stage. The precondition of this stage is that the radio should be aware of possible alternative actions. The following list includes some of the possible actions for this stage:

- **Reconfiguring existing applications:** This action includes determining the necessary parameter changes of the running applications. Reconfiguring some of the parameters can be harder compared to others. This depends on the flexibility of the component that the action affects.
- **Switching between applications:** In this action, the CR decides to switch or not, between installed applications. Installing multiple applications may require

a radio to have enough memory, processing power, and antennas that are required for the applications.

- **Installing new applications:** This action is the process of deciding whether to install additional applications or not. This may require over-the-air downloading of new applications or standards.

2.2.3.4. Acting stage:. Acting is the final stage of the cognitive life cycle. In this section, the decisions of the previous sections are gathered and the required reactions are performed by the CR. These reactions affect both the internal conditions and the external conditions of the radio. Therefore, this step also triggers the first step which is sensing stage so that it will sense updated conditions in the next turn.

Since these four stages behaves cascadingly, that is they affect each other, the time frame of a single cycle is an important parameter. If it is too small, then the first step may start again before the last one completes, on the other hand if it is too long then the CR cannot react simultaneously to the environment changes.

2.3. Design Patterns

In order to design a SDR that satisfies the requirements of CR in terms of portability, reconfigurability, and reusability, it is imperative to be familiar with software engineering concepts such as object oriented programming (OOP) [90] and design patterns [91]. The design patterns constitute optimum solution to common software engineering problems. Applying design patterns can speed up the development process by providing tested and proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Using design patterns helps to prevent subtle issues that can cause major problems. It also improves code readability for coders and architects who are familiar with the patterns.

Design patterns can help people learn object oriented thinking: leveraging polymorphism, designing for composition, balancing responsibilities, and providing plug-

gable behavior. The importance of using suitable software design patterns is presented in more detail in [90]. The authors present 23 design patterns organized into three categories [90, 92, 93]:

2.3.1. Creational Patterns

Creational patterns deal with object instantiation problems. There are 5 creational patterns including Abstract factory, Builder, Factory, Prototype, and Singleton. They are intended to solve object creation problem.

Creational patterns can be explained briefly as follows:

- **Abstract factory:** Groups common factories that are used to create objects that implements a common interface.
- **Builder:** Simplifies creating complex objects by defining sub steps that separates construction and representation.
- **Factory:** Defines a factory class to create objects that implements a common interface. Created objects can be used by predefined interface.
- **Prototype:** Helps to create similar objects by cloning an existing base object.
- **Singleton:** Ensures that a class can be instantiated only once.

2.3.2. Structural Patterns

Structural patterns consist of Adapter, Bridge, Composite, Decorator, Facade, Flyweight, and Proxy. These patterns concern class and object composition. They use inheritance to compose interfaces and define ways to compose objects to obtain new functionality. These patterns may be applied while designing a new system from scratch or while modifying existing codes to port from one system to another.

Structural patterns can be explained briefly as follows:

- **Adapter:** Allows classes without compatible interfaces to work together. It

adapts one interface to the other.

- **Bridge:** Separates an abstraction from its implementation so that they can be decoupled.
- **Composite:** Lets clients to behave individual objects and composition of objects in the same manner.
- **Decorator:** Dynamically adds new functionality to an existing class.
- **Facade:** Defines a simplified interface to a complex implementation.
- **Flyweight:** Reduces the memory consumption of creating large number of similar objects by sharing common data.
- **Proxy:** Helps controlling access, reducing complexity, and minimizing cost for a complex object by placing it in simpler proxy objects.

2.3.3. Behavioral Patterns

Behavioral patterns including Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template method, and Visitor, solve object communication problems and depicts how objects act together.

Behavioral patterns can be explained briefly as follows:

- **Chain of responsibility:** Chains the objects and lets more than one object to handle a single request.
- **Command:** Creates objects to represent actions.
- **Interpreter:** Allows to define new grammars.
- **Iterator:** Defines an access method to objects without caring about their data representation.
- **Mediator:** Creates a manager class that defines the interaction of other classes and their methods.
- **Memento:** Keeps the previous state of an object and allows to restore it.
- **Observer:** Notifies a group of objects whenever a change occurs.
- **State:** Localizes state dependent code to state specific classes.
- **Strategy:** Provides the ability to choose among set of algorithms at runtime.

- **Template method:** Allows to define the steps of a large algorithm in subclasses.
- **Visitor:** Adds virtual functions to a class without modifying the existing codes.

All of these patterns help developers to design software better in terms of quality, reusability, and understandability. Basically, all of the design patterns tell the following golden rules in common[94]:

- All client objects should always call the abstraction (interface) instead of the exact implementation.
- Future changes should not require modification on the existing system.
- Only changing parts should be modified.
- Objects should be loosely coupled.

2.4. Middleware

Special techniques can be used to decrease the dependency between different software types or between components of a software. Using middleware between software entities is the most prominent method. The term middleware can be defined as "the software layer that lies between the operating system and applications on each side of a distributed computing system in a network" [95]. In other words, the middleware is simply the software in the middle between two layers.

The reason behind using middleware in a system design may vary according to the type of the application such as messaging, invoking remote methods, identification, distributed computing, authentication over network, trading, security, database operations, and so on. The typical purpose of using middleware is to provide hardware and software transparency to the software components. In the next subsections, the advantages and disadvantages of using middleware in software design is discussed and some of the common middleware types are summarized.

2.4.1. Advantages of Using Middleware

The advantages of using middleware while designing a system can be listed as follows:

- Middleware technologies may increase portability of the application as long as the middleware can be ported to target platforms. It means that limitation of the portability becomes the portability level of the middleware if all other portability requirements are satisfied. This can be ignored if portability is not a key issue for an application.
- It can reduce complexity by separating application code with the platform dependent code. The platform dependent code goes into the middleware so that the developer can focus on the functional parts of the application which yields more efficient time usage.
- It can increase flexibility of the system by allowing to insert or remove components between two different sides without affecting the already working components.
- It can increase reusability of the software components. Since the software is forced to be divided into components and the components talks each other over predefined interfaces, they can be reused in other software the requires the same interface. It decreases the development time and the effort of the developers. In addition, reusability can reduce implementation errors by allowing to use tested and proven reusable components.
- It allows implementation to run distributedly among different platforms. This is a big advantage if the application logic requires running with server and client strategy, because otherwise the developer hast to implement all network logic.
- Using middleware may help to reduce development cost by supporting additional platforms with one version of the code.
- Integration of new and legacy components may be handled more easily.

2.4.2. Disadvantages of Using Middleware

The disadvantages of using middleware while designing a system can be listed as follows:

- It increases latency between parts of the software and makes components to work slowly compared to non-distributed implementations of the same application.
- It makes the system more complex consisting of multiple parts.
- Any component that does not support the middleware fails to work with the rest of the system.
- Implementation becomes middleware dependent and does not work on a system that does not support middleware. It means if it is not possible to install the middleware onto a platform, then it is also not possible to install the application to that platform.
- Reliability of the software may decrease due to the increasing number of network layers. It may affect the systems where real-time acting is crucial.

2.4.3. Classification of Middleware Technologies

The middleware technologies can be classified based on scalability and recoverability as follows [96]:

- **Remote Procedure Call:** In this type of middleware, the client can make asynchronous or synchronous calls to procedures running on remote systems such as other computers on a shared network. In synchronous calls, the clients waits for response from the server in a blocking manner. On the other hand, in asynchronous remote procedure calls, the client does not wait for the response and continues to work. The server returns the response back to the client when it completes the task.
- **Message Oriented Middleware:** In message oriented middleware, while the client continues with other processing, it collects and stores the incoming messages until they are processed. Most message oriented middleware depend on

message queues, but there are also some broadcast- and multicast-based messaging systems.

- **Object Request Broker:** This type of middleware allows clients and servers to talk to each other over a predefined interface. CORBA is an example of object request broker middleware. CORBA, as a middleware supports multiple programming languages and multiple execution platforms. This flexibility makes CORBA systems candidate platforms for heterogeneous system such as SDRs. CORBA is mentioned in more detail in the following section.
- **SQL-oriented Data Access:** This middleware allows applications to access database servers remotely.
- **Transaction processing monitors:** Transaction processing monitors are one of the first middleware and are usually used in three tier applications. The main purpose of this middleware is to monitor the transactions to complete successfully. In case of failure the transaction processing monitor takes appropriate actions vital for commercial business world.
- **Application servers:** Application servers are commonly used to isolate the business logic with the rest of the system. They usually provide services such as transaction management and load balancing.
- **Enterprise Service Bus:** It is an abstraction layer on top of an Enterprise Messaging System.

2.4.4. CORBA

This section is dedicated to CORBA since it constitutes the best candidate platform for SDR. The underlying motivation in the development of Common Object Request Broker Architecture (CORBA) [66] is the ability for any client to talk to any server in the same network. In fact, CORBA is intended to solve the communication problems of heterogeneous systems. CORBA provides asynchronous and synchronous, loosely-coupled timeless communication between software entities. CORBA standard is mandated by Object Management Group (OMG), an international, open membership, non-for-profit computer industry consortium [65].

CORBA hides the actual communication mechanisms under an Object Request Broker (ORB). It also assists in development of distributed applications between multiple processors, hides object communication issues, provides common services, and automates common networking tasks.

Code development using CORBA starts by defining the interfaces. An interface is the contract between a client object and a server object. It can also be thought to be the textual representation of the system design and is defined by a C like language called Interface Definition Language (IDL). IDL basically defines attributes, types, operations, and the parameters of the operations. IDL allows defining remote methods that accept parameters in a wide range of complexity from primitive types to complex objects.

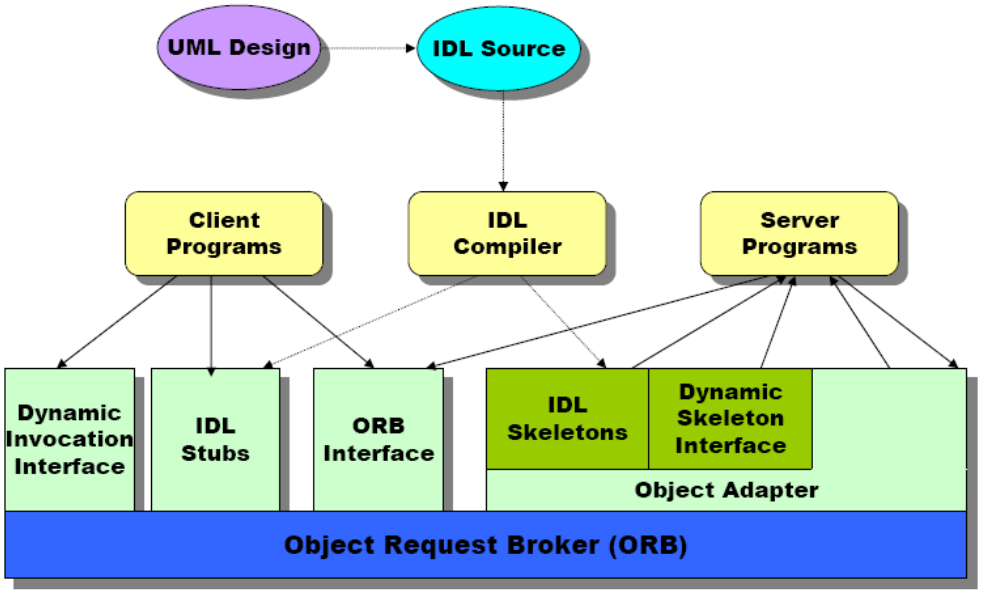

Figure 2.9. CORBA Development [97]

The development diagram of a typical CORBA program is shown in Figure 2.9. Predefined system design such as Unified Modeling Language (UML) may be used to generate Interface Definition Language (IDL) file. After determining the interface, the IDL file is compiled by an ORB vendor provided IDL compiler for each implementation language to generate stubs and skeletons. The automatically generated stub and skeleton files contain the networking details and they are not modified by the devel-

oper. After that step, the developer implements the IDL operations in a programming language such as C++, Java or any ORB supported language. Figure 2.10 shows the CORBA communication among different processes.

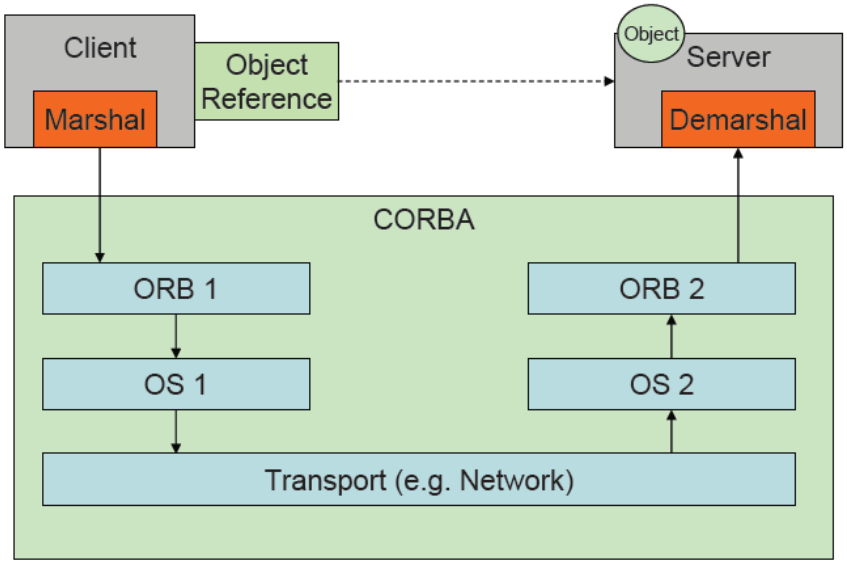

Figure 2.10. CORBA Communication Among Different Processes

The lifecycle of a CORBA call starts with the client object obtaining the object reference of the server. The network address of the server object, which is called an Interoperable Object Reference (IOR) may be obtained manually, or by searching among pre-registered addresses in the naming service of CORBA. Naming service works like a phone book where server objects register themselves for clients to locate them. It allows location transparency between CORBA objects. After obtaining the object reference, the client invokes server operation via stubs. The client ORB converts the function call to a CORBA request and transfers the request to server ORB. Server ORB converts CORBA request back to function call, associates function call with servant implementation, and finally dispatches the request. Once a client obtains a remote reference, it can use the reference as easy as using a local reference, since all the networking details are handled by the ORB.

2.5. Configuration Management

Configuration management [98] is the task of managing and controlling parameters of the software. It can be a vital problem in large systems where tracking subsystems is a trouble. In a software system, each subsystem may have dynamic or fixed parameters. In fixed configurations, usually the parameters are embedded in the code, thus changing a parameter requires recompilation. In simple systems where the number of adjustable parameters is small this may be a short cut solution. However, in large and frequently changing systems an external configuration management is a must. Configuration management system can be as simple as a text file or can be a complex remotely connected database system. The capabilities of the target platform constitute a key point in the design of efficient configuration management. If the system has a powerful processor, using complex configuration management systems becomes an option. If the resources are limited and portability is an issue text based configuration systems such as eXtensible Markup Language (XML) can be considered.

2.5.1. XML

The word "extensible" in the open form of XML refers to the fact that it is a general-purpose markup language for creating custom markup languages [99]. As the extensible word refers it is a general-purpose markup language for creating custom markup languages. The ability of users to define their own elements makes XML a candidate for configuration management. The purpose of XML is to help information systems to keep their data in a structured way. The lexical grammar and the requirements for parsing of XML files are published by the World Wide Web Consortium (W3C) [100]. It is a free and open standard. Figure 2.11 shows a sample XML document.

```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology
      society in England, the young survivors lay the
      foundation for a new society.</description>
   </book>
</catalog>
```

Figure 2.11. Sample XML Document

The advantages of using XML while designing a system can be listed as follows:

- It is relatively human-legible text file.
- It uses Unicode so it supports many languages.
- It can represent common computer data structures: records, lists, and trees.
- Its intuitive format describes structure and field names as well as specific values.
- The pre-defined syntax and parsing requirements make the necessary parsing algorithms much more simple, efficient, and consistent.
- XML can be used online and offline as a format for document storage and processing.
- It is based on international and open standards.
- It is a scalable language.
- It allows validation using schema languages such as DTD, XSD, and Schematron, which allow effective document checking against errors.
- The hierarchical structure supports most formats.
- It is platform independent.
- It is forward and backward compatible to maintain changes in DTD or Schema.

The disadvantages of using XML while designing a system can be listed as follows:

- The syntax of XML is redundant (Each open tag has a close tag) or larger in size compared to binary representations of similar data.
- The redundancy causes higher storage which affects application efficiency by transmission and processing costs.
- XML syntax is verbose, that is it contains unnecessary keywords compared to other text based configuration management systems.
- Overlapping (non-hierarchical) node relationships require, extra effort to explain.
- It is problematic to use namespaces in XML, and parsing namespaces is difficult.
- The distinction between content and attributes is unnatural to read and makes XML data structures harder to design.
- Transformations usually result in changes of format such as whitespace, newlines, attribute ordering, and attribute quoting. These problems can make it very difficult to compare XML source differences among several source files.
- XML contains non-relational (non-normalized) data structures.

3. PROPOSED CHANNEL SELECTION ALGORITHMS

As explained in previous chapters, CR aims to increase efficient utilization of the unused spectrum during operation. This is achieved by active monitoring of the RF environment and user activities in that environment. There are several parameters that can be optimized to achieve most efficient spectrum usage. They have been summarized in previous chapters. As a consequence of that optimization process, CR alters its transmission and reception parameters to dynamically use all parts of the available spectrum.

In this chapter, we focus on the optimization of channel selection procedure to minimize the number of handoffs. Handoff is the process of transferring an ongoing call or data session from one channel to another channel. It is the main argument of the CR to provide dynamism. However handoff is a costly process. The reasons of that can be listed as follows:

- Deciding whether a handoff process is required or not is a hard question to answer, since it requires a lot of processing power in terms of spectrum monitoring, signal intelligence, and interference detection. Timing for the handoff process should be decided by sensing external environment precisely. Because wrong handoff causes communication failures for all of the devices affected from handoff process. Therefore, CR should be aware of the RF environment and should be able to detect primary and secondary users in that environment in order to prevent communication interference. In addition, all of these processes should be performed in the shortest time in order to prevent performance problems for the ongoing calls.
- Handoff process requires the CR to monitor device capabilities at run-time so that CR can decide whether it can reconfigure itself for the new conditions or not.
- Since handoff requires active sensing, there is also a trade-off between the number of handoffs and battery consumption. This can be a problem for mobile handsets

where battery is critical.

It is important for the CR to minimize the number of handoffs during a connections. It should always aim to complete its connection with the same parameters as much as possible.

In this chapter, we propose two channel selection algorithms to minimize number of handoffs for the CR: Average Holding time channel Selection algorithm (AHS) and Probabilistic Channel Selection algorithm (PCS). We define a channel model where these algorithms are implemented and discuss the algorithms briefly in the next sections.

3.1. Channel Model

In this section, we describe our channel model as a starting point. Figure 3.1 shows an example representation of our model.

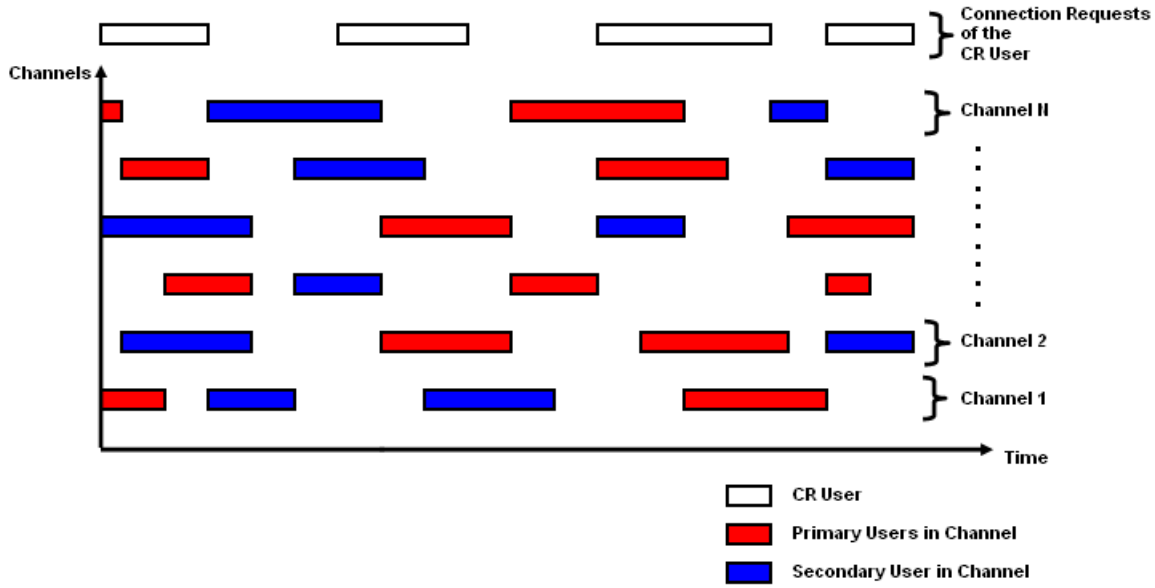

Figure 3.1. Channel Model

As shown in the figure, spectrum is divided into several channels where each channel has its dedicated transmission properties such as frequency or modulation. In addition, each channel is composed of channel users and unused spectrum. Channel

users are classified by primary and secondary users. In the figure, red boxes represent channel usage duration of the primary users and blue boxes represent channel usage duration of the secondary users. White area between these boxes represents idle spectrum which is not utilized by any channel user at that time.

We add our CR user to this model and run our algorithms on it to optimize its handoff performance. Our CR user tries to perform connections represented by the white boxes at the indicated time. Left side of the white box means that user starts a new connection and right side of the white box means that user ends the connection. When a user connection request arrives and when user has to make a handoff it performs the required operations to select an available channel.

We can summarize the underlying assumptions in the development of our model as follows:

- RF environment is divided into identical channels where the number of channels is at least 1.
- There exists only secondary and primary users in these channels.
- All primary and secondary users are assumed to be identical.
- Channel users may start transmission at any time on an available channel.
- Primary users are the main owners of the channels. They have precedence of using a channel over secondary users.
- Our CR user is also a secondary user and it is capable of using all channels as long as they are available.
- All device related and environmental errors are ignored in order to simplify the model.
- User specific preferences and pricing models for handoff are ignored.

3.2. Channel Selection Procedure

In this section, channel selection procedure is explained. Our CR device performs the following channel selection procedure when it starts a new connection or it has to make a handoff during an ongoing connection.

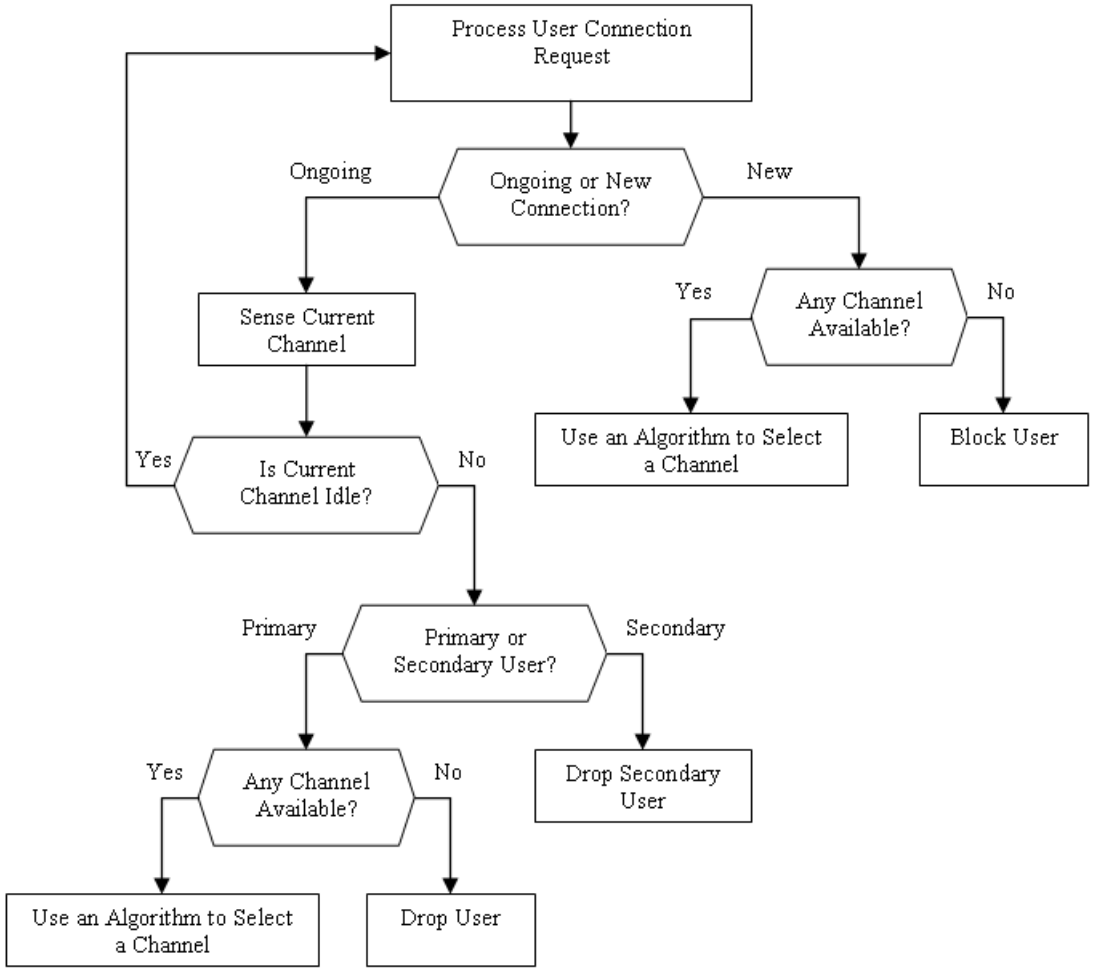

Figure 3.2. Channel Selection Procedure

Figure 3.2 shows the flow diagram of our channel selection procedure. It can be explained as follows:

1. Channel selection starts when a user connection request arrives.
2. CR determines that whether this is a new connection request or an ongoing connection which is already started.
   (a) If this a new connection request, CR uses an algorithm to select an available

channel.

i. If all channels are used by other users and there is no available channel then CR is blocked.

ii. If there exists at least one channel CR uses an algorithm to select a channel.

(b) If this an ongoing connection request, CR senses current channel and checks whether any other user exists. If a user is detected than CR determines whether its a primary user or a secondary user.

i. If a primary user is detected in channel, CR searches for alternative channels.

- If there exists at least one channel, CR uses an algorithm to select a channel.
- If all channels are used by other users and there is no available channel then CR is dropped.

ii. If a secondary user is detected in channel, it's connection is dropped and CR continues to communicate.

Figure 3.2 shows the state diagram of the CR during its operation. As shown in the figure, CR may be in one of the five different states.

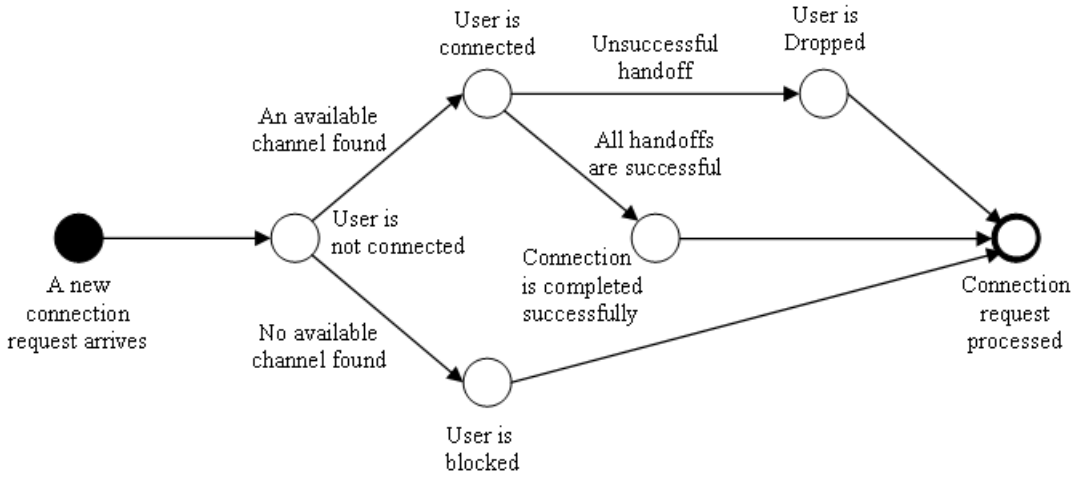

Figure 3.3. State Machine of Processing a User Connection Request

- **Not Connected:** This is the initial state for CR and this states indicates that CR does not have any connection request at the moment.

- **Connected:** This state shows that CR has successfully selected a channel from available channels and it is communicating now.
- **Blocked:** At the beginning of a connection request of the CR, it selects an available channel to start communication. If there is no available channel, then CR is blocked and it's connection request fails.
- **Dropped:** During operation of the CR, if a primary user starts transmitting at the selected channel, then CR tries to select another available channel. At that time, if CR cannot find any available channel, then it's connection is dropped.
- **Connection is Completed:** This state indicates that, CR has completed it's connection request successfully. In other words, no blocking or dropping has occurred and all handoff attempts were successful during operation.

3.3. Average Holding Time Channel Selection Algorithm (AHS)

In this section, we propose an algorithm to minimize the number of handoffs of CR during its operation. AHS algorithm aims to select a channel when a new call request arrives or a handoff is required. The idea behind the algorithm depends on fitting user holding time average to the blank time average of the channels. Therefore the algorithm is called AHS. The steps of the algorithm can be explained as follows:

1. Calculate average holding time of the user (h_u) up to current time.
2. Calculate average blank time of each channel (w_c) up to current time.
3. Select best fitting channel as the following:
   (a) If some of w_c is equal to h_u then select the channel which is found first.
   (b) If any of w_c is larger than h_u then select channel that has the smallest w_c.
   (c) If none of w_c is larger than h_u then select channel that has the biggest w_c.

In this algorithm, user holding time average is calculated by dividing the sum of user holding times to the number of calls. Similarly, average blank time of each channel is calculated by dividing the total blank time of each channel to the number of blanks in that channels.

It must be noted that AHS algorithm selects the candidate channel by considering the average holding time of the user. In other words, it does not select the channel which has the maximum blank time average, but instead compares the blank time average of the channels with the user holding time average and determines the channel as a result of that comparison. The idea behind this approach is based on using the channels in the most efficient way and distributing channels among CR users proportional with their needs.

To illustrate the AHS algorithm, suppose that at time t, CR has the following variables:

- User holding time average is calculated as 35 seconds.

- There are 6 channels with the following channel blank time averages:
  - Channel 1: 10 seconds
  - Channel 2: 20 seconds
  - Channel 3: 30 seconds
  - Channel 4: 40 seconds
  - Channel 5: 50 seconds
  - Channel 6: 60 seconds

In this scenario, AHS starts with searching for a channel with average blank time of 35 seconds. Since there is no channel with average blank time of 35 seconds than it searches for the channels with larger blank time average and it selects channel 4 which has the smallest channel blank time average among other channels with the blank time average of larger than 35 seconds. If there had been no channel of which blank time average is larger than 35 seconds, AHS algorithm would select channel 3 of which channel blank time average is the biggest one among other alternative channels.

3.4. Probabilistic Channel Selection Algorithm (PCS)

PCS algorithm is our second proposed algorithm for minimizing the number of handoffs during channel selection. It is different from AHS in that it selects the channel in a probabilistic manner instead of calculating average holding time. The steps of the PCS algorithm can be explained as follows:

1. Calculate average holding time of the user (h_u) up to current time.
2. Calculate fitting probability of h_u to each selectable channel as follows:
   (a) Calculate the number of fitting of h_u to each selectable channel.
   (b) Calculate probability of fitting as the rate of number of fitting to number of tries.
3. Select channel that has the maximum probability.

PCS algorithm is similar to AHS algorithm in that CR starts with calculating average holding time of the user up to current time. In the second step, PCS algorithm

calculates fitting probabilities of each channel. Fitting means that channel blank time is greater than or equal to the user holding time average. In other words, if the user can complete his connection without handoff in a channel, this is called a fitting. Number of fitting is calculated by counting the number of fittings of user holding time average to the channels up to the current time. After this step, PCS algorithm selects the channel with the maximum probability.

4. SDR DESIGN AND IMPLEMENTATION

In this chapter, we design and implement an SDR to realize our proposed channel selection algorithms on a real platform instead of implementing only the simulation tools. Our design has several layers including hardware, operating system, core framework, CORBA, and waveform. We present these layers and the components in each layer. We use an x86 based PC as our hardware, Pardus 2008 as our operating system, SCARI-Open as our core framework, and ACETAO for our CORBA layer. We design and implement a proof-of-concept waveform in C++ for our SDR platform. We present the design and implementation steps of our waveform application. We also summarize the tools we have used in each development stage. Finally, we propose some design patterns to extend our architecture.

4.1. Big Picture

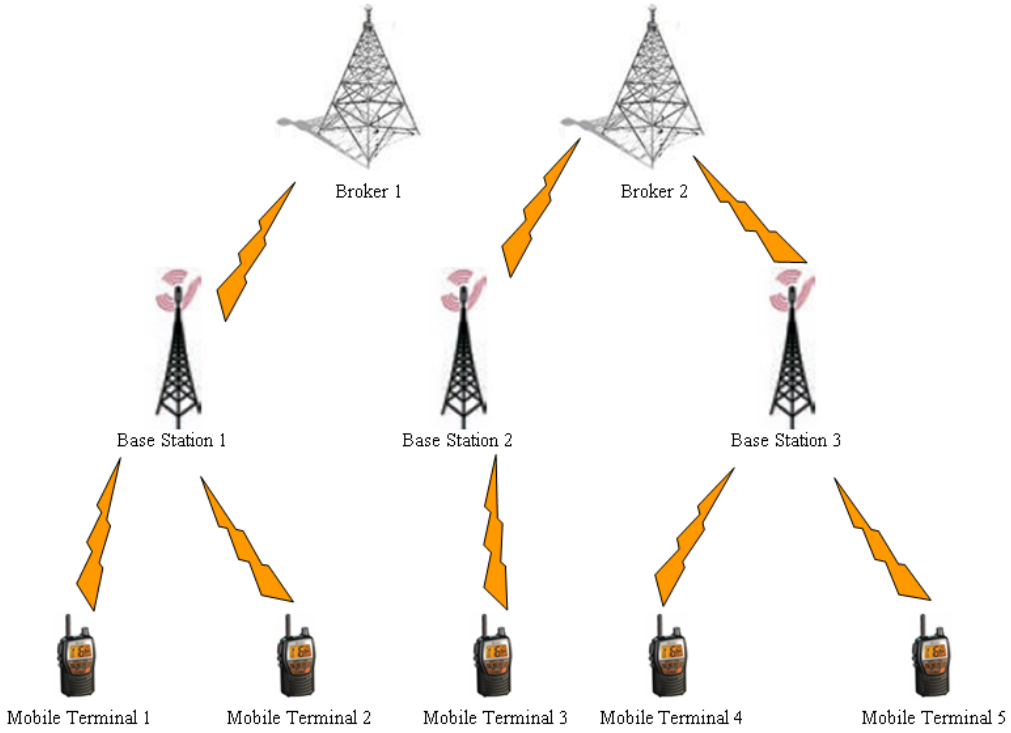

Figure 4.1. Testbed Infrastructure for CR Network

This thesis is a part of a CR testbed development effort in Bogazici University.

In the testbed, we aim to simulate a CR network infrastructure [101]. Figure 4.1 shows the elements of our CR network and their relationship.

The testbed has three main actors:

- **Mobile Terminal:** Mobile terminal is the CR device which is based on SDR architecture. Each mobile terminal has a unique ID(Such as GSM no) and a frequency value assigned from base station. It also has a quality of service (QoS) parameter which is used to select the base station it is connected. A mobile terminal can change its frequency or base station according to its QoS parameter. Each mobile terminal has a waveform installed. The waveform application has two modes: Voice and Data. Each mobile terminal can talk to another mobile terminal or send a text or binary file to another mobile terminal.
- **Base Station:** Base station works as a gateway for mobile terminals. Each base station has a frequency pool and can assign a random frequency from its pool to the connected mobile terminals.
- **Broker:** Broker manages the base stations and assigns them a sub frequency pool.

In this thesis, we focus on the design of the mobile terminals. The other actors of the testbed is out of scope for this study. We propose an SDR design and implement the waveform of the mobile terminals. The waveform application runs cognitively that can understand the type of the incoming data and behaves accordingly. If the incoming data is voice it plays the data or if the incoming data is a text or binary file then it writes to the disk with a file name containing the ID of the sender. In addition, our waveform application can send the data with a QoS parameter by selecting the most appropriate base station and frequency. In order to simplify the scenario, we use personal computers to simulate real mobile terminals. They are all connected to each other over network connection instead of real antennas.

The software architecture of SDR based mobile terminals has several layers which are shown in figure 4.2.

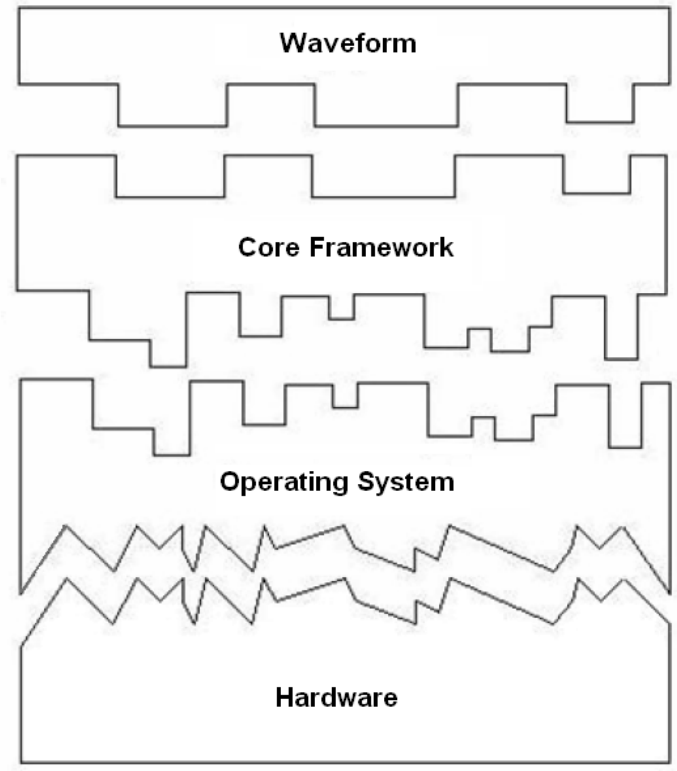

Figure 4.2. Layers of a Mobile Terminal

The layers of the mobile terminal and their functionalities can be explained as follows:

- **Hardware:** The hardware layer is a PC which has at least the following components:
  - **GPP:** It is the processor of the system. In our development environment we use a 2 GHz Intel based processor.
  - **Sound Card:** Since our waveform can support voice communication, the PC must have a sound card with its driver installed.
  - **Ethernet:** Ethernet is used to send or receive data to other nodes over the network. We use ethernet connection instead of real RF communication to simplify the scenario.
  - **Memory:** We use 20 GB of hard disk partition dedicated to the operating

system and other applications and 2 GB of RAM in our testbed.

- **Operating System:** We use Pardus as our operating system. Pardus is a Linux based operating system which is developed by TUBITAK UEKAE.
- **Core Framework:** Core framework provides a standardized layer to manage the waveform applications. It can install, configure, start, stop, release and uninstall a waveform. The basic task of the core framework is to read the configuration files of the waveform and deploy the waveform executables to processors that meets the dependencies, create CORBA connections among them, and configure the waveform components according to the XML files of the waveform. Another very important task of the core framework is to provide standardized interfaces for hardware devices and platform services to the waveforms. Therefore, the waveforms can use the devices of the platform without caring about technical details.

  We use SCARI-Open core framework of Canadian Research Center (CRC) for this layer. It is a Java based open source core framework and supports Linux PC platforms. SCARI-Open core framework provides the following device and service implementations that can be used by waveforms:
  - **AudioDevice:** AudioDevice provides an abstraction mechanism for sound card. In our waveform implementation we connect to the AudioDevice of SCARI and capture sound by using it.
  - **ExecutableDevice:** It provides an generic interface for platform processor. The executable files of our waveform are loaded and executed by that device of the core framework. This device is responsible to execute and terminate waveform components.
  - **RFDevice:** SCARI-Open provides an RF device implementation however it can only be used with an RF device hardware of CRC. At boot up SCARI checks for physical RF device to be connected to RS232 port of PC and if it is not connected, SCARI does not start this logical device.

    In our implementation we do not use SCARI RFDevice since we do not own that hardware and also we do not use real RF communication but instead we implement our own logical RFDevice component to use ethernet

connection. The waveform does not care about how the packets are really sent. As in the case of SCARI, our RFDevice implementation provides an abstraction mechanism for network operations and it can be easy to modify our implementation to send the packets over the air in the future.

- **LogService:** It provides a logging mechanism for waveform components. Once the waveform components connect this service over CORBA, they can call appropriate methods to produce logs.
- **FileSystem:** File operations are also implemented by the core framework and provided as CORBA interfaces to waveforms so that the waveforms can make file operations regardless of the underlying operating system and hardware. SCARI-Open does only provide a file system for Linux operating system, however it is possible to implement a file system to provide access the files on embedded devices such as a flash memory on a mobile phone.

  The waveform can treat every file system in the same manner because of unique interface. Because of CORBA, it can call the file operations even the physical locations of the files are distributed. The waveform can save read or modify a file which is physically located on another PC over a network. All of these mechanisms increase waveform portability.

- **Waveform:** The last layer of our mobile terminal is the waveform. Waveform is the main application of the mobile terminal and it implements the actual operations of the radio. a SDR device can have multiple waveforms at the same time. The structure of the waveform layer is detailed in the next sections.

4.2. Waveform Design

In this section we present the design details of our waveform application by providing design models and component layout. We also mention about the packet structures and component interactions among each other.

The waveform that we want to design will have the following functionalities:

- Each mobile terminal will have a unique mobile ID which has to be reconfigurable.
- A mobile terminal will be able to communicate with another mobile terminal by calling its mobile ID.
- Mobile terminals will support voice and data communication. In voice mode, the mobile terminal will be able to talk to another mobile terminal and in data mode it will be able to send a text or binary file to another mobile terminal. Mobile terminal will be cognitive to understand the type of communication and behave accordingly when receiving incoming data. If it is a voice communication, the mobile terminal will play the incoming data and if it is a data communication then mobile terminal will save it to a file with the mobile ID of sender.
- Communication will have a reconfigurable quality of service criteria between 1 and 5 where 1 corresponds to worst and 5 corresponds to best quality level. It can be assumed that there is a tradeoff between cost of communication and the quality of service parameter in this scenario.
- The waveform application will support dynamic installation and uninstallation to mobile terminal at runtime.
- The mobile terminal will provide a user interface to configure its reconfigurable parameters at runtime.

When designing our waveform, we keep our metrics in mind to maximize:

- Portability
- Scalability
- Reusability
- Reconfigurability
- Upgradability

It is possible to add more items to the list above. As a starting point, in order to achieve our goals we follow "*divide and conquer*" methodology when designing our system. The reason is dividing a big problem into smaller sub problems allows to

design and implement the whole system with less effort. Also smaller components bring scalability to the system by adding new components and removing existing ones. It is also easier to upgrade or modify the system by only changing sub units.

Determining the granularity of the system is another question mark. In other words, we intend to divide our waveform into sub units, but the question is "How many components shall we create?". The key idea behind determining the components of the waveform has several aspects:

- The first one is the functionality aspect and it means that each component shall have a different mission from each other and it should focus on to achieve it. It is up to the developer to determine the separation level of the missions. However, it should be neither too small, nor too complex.
- Another aspect is the reusability level of the components. When designing the waveform, we must remember that each waveform has some similar parts. For example, capturing audio and playing audio are a general functionalities that every audio waveform should support, also every waveform has some file operations such as reading and writing files. Since our waveform will have both audio and file operation, we can design these functionalities as a separate component, in order to be able to reuse them when designing new waveforms.
- In addition the items above, the developer should keep in mind to determine dependencies of the waveform component. Since SDRs may have a different processing units such as GPP, DSP and FPGA the target processing platform of the jobs must be considered in mind when determining the granularity. Table 4.1 shows the common deployment strategies of SDR jobs according to the processing units. Since we use a PC platform, we have only a GPP and we will deploy our components to the GPP of our PC so that we can neglect this criteria when designing our waveform.

Table 4.1. General Deployment Strategies of the SDR Jobs According to Processor

FPGA	DSP	GPP
Digital up/down conv.	Resampling	Resampling
Equalization	Carrier sync.	Carrier sync.
Eq. and pre-emp. filt.	Carrier sync.	Carrier sync.
Chip rate/code sync.	Symbol sync.	Symbol sync.
Spread/despread	Modulation/demodulation	Modulation/demodulation
Carrier Synch.	Interleaving/De-interleaving	Interleaving/De-interleaving
Symbol Synch.	Packet framing	Packet framing
Diversity Combining	Error correction	Error Correction
Resampling		Error correction

Figure 4.3 shows the model of our waveform application. Our application has eight components. The model shows these components with large boxes. The lines connecting the boxes represent the CORBA connections among components. For example, ReadData component is connected to ProcessTx component over CORBA which means ReadData will send some data to ProcessTx component by using a predefined CORBA interface. The line representing CORBA connection has two small boxes at the border of components. These represent the Port objects. A Port object is the CORBA object implementing Port interface. Notice that one of the port object is black and the other is white. The color represents the client and the server. White means that this port is the server port and it implements the operation and black means that this is the client object and it wants to call the operation on the server object. Except from the connections between waveform components, there are some floating small rings or boxes that the components have connections. They represent a component of the core framework, or an object that can be found by using CORBA naming service. For example, PlayData has a connection to the AudioIn floating port. It means that PlayData waveform component, connects to the AudioIn port which is located in AudioDevice of the core framework. PlayData uses that port to play the incoming data by using the device of the core framework.

Each waveform component has a corresponding executable file and some config-

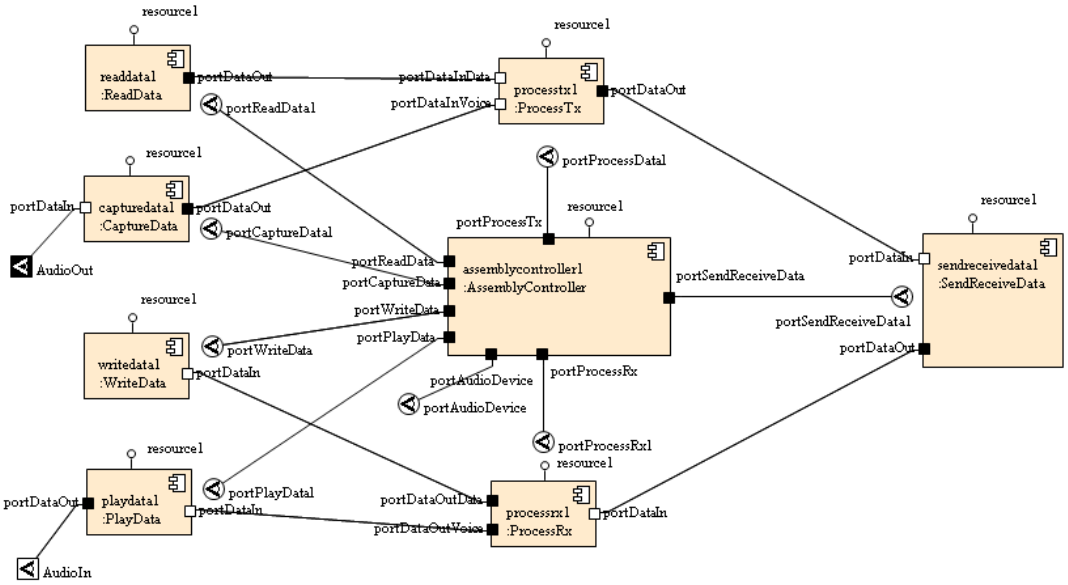

Figure 4.3. Zeligsoft Model of the Waveform

uration XML files. XML files mainly describe the following:

- **Connections:** Connections are defined in XML files and they describe "Which component talks to which component?" Modifying XML files can change the connections among different components. As far as two components have the same interface, they can be connected to each other. Therefore, it becomes easy to create new applications by connecting different components. The connections are established by the core framework when installing the waveform.
- **Dependencies:** It defines the requirements to run the executable file of the component such as the operating system that the executable file is compiled for, processor type to run this component, memory requirements and so on.
- **Configurations:** Configurations defines reconfigurable properties of each component. Each waveform component is initially configured with the value written in configuration XML file. The configurations can be changed by using PropertySet interface over the CORBA at run time.

The functionalities of each waveform component are summarized below:

- **ReadData:** ReadData components is responsible to read the input data file

as a sequence of bytes and send them to the ProcessTx component for further processing. The number of bytes to be read is defined as a configuration parameter and it is currently set to 8192 bytes at each step.

- **CaptureData:** CaptureData component is the input source component of the waveform when running in voice mode. It is connected to the AudioDevice component of the core framework. Once it is connected, the AudioDevice component of core framework captures data from physical audio device and sends the voice as a sequence of integers to the CaptureData component. It must be noted that CaptureData does not deal with the physical hardware and the operating system to capture voice. The layered architectural design, lets the application to be independent of the rest of the system as far as the CORBA connections are established. CaptureData component sends the captured voice to the ProcessTx component for next operations.
- **WriteData:** As the name refers, this component writes the given data array to a file. This component extracts the ID of the sender from incoming packets and creates a file with the name of sender. It supports both text and binary files.
- **PlayData:** This component is responsible to play the given voice data. As in the case of CaptureData component, PlayData does not directly connect to the physical audio device of the radio, but instead uses AudioDevice component of core framework to play the given sound. AudioDevice is responsible from the capacity values of physical audio device and if the capacity is full, it does not allow more connections. In addition, it provides a central initialization and configuration mechanism for sound card, therefore multiple initializations are prevented and once the AudioDevice configuration is changed all the radio is affected.
- **ProcessTx:** ProcessTx component prepares the packet to be sent over network. It appends a header to the message. Figure 4.5 shows the content of a packet between the waveform and the RFDevice. The RFDevice also adds some additional header to the packets. After packetizing it sends the packets to the SendReceiveData component.
- **ProcessRx:** ProcessRx component unpacks the packets and extracts the header and data from them. It also reads the header and analyzes the incoming data.

If it is a voice packet, this component forwards the data part of the packet to PlayData component, and if the incoming packet contains binary or text file data than the incoming packet is sent to WriteData component to be written to the disk.

- **SendReceiveData:** This component has two responsibilities: send packets coming from ProcessTx component to the RFDevice and receive packets from RFDevice and send them to ProcessRx component.
- **AssemblyController:** AssemblyController is the manager component of the waveform and it controls the assembly of the waveform component. It is responsible to talk to the core framework and forward the incoming commands or configurations to other components. Each waveform should have a predefined AssemblyController component which is specified in the XML configuration files of the waveform. Once the core framework reads the configuration files, then it talks to AssemblyController component for any managing operations. For example, in order to start or stop the waveform, core framework calls start or stop method of the AssemblyController and AssemblyController calls start and stop on the rest of the components. The main idea behind this architecture is to make any waveform application transparent to the underlying core framework. Since core frameworks know that there is an AssemblyController component in the waveform and it can use that component to manage that waveform regardless of the internal complexity of it.

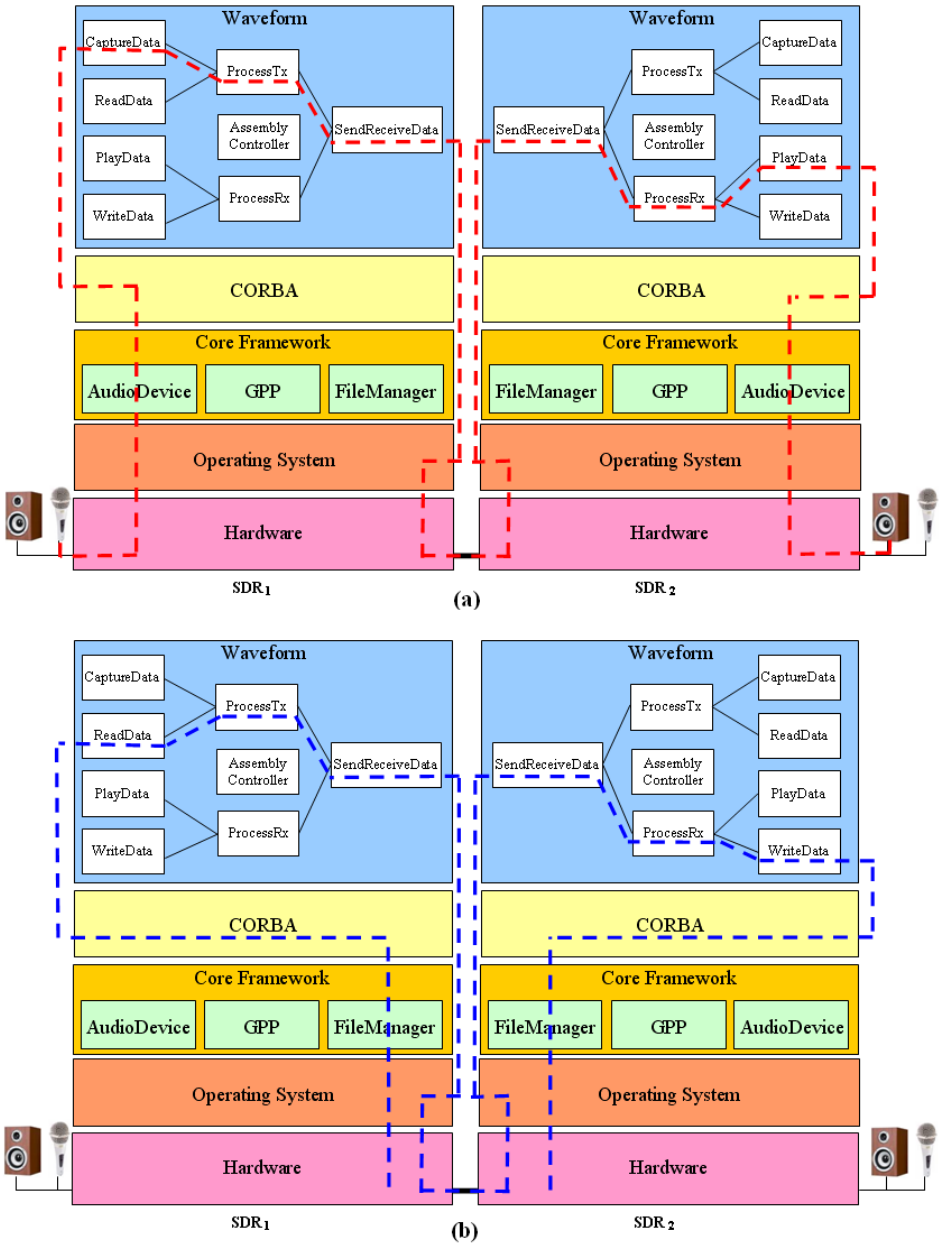

Figure 4.4. Example scenario where two SDRs communicate (a) Path of the packets in voice mode (b) Path of the packets in data mode

These components make up the waveform and once the waveform is started they begin to transfer packets to each other along CORBA connections. Figure 4.4 shows an example scenario where two SDRs communicate in our proposed design. It also shows the layers of our design and their interaction with each other. Red dashes represent the path of the packets in voice mode, and similarly blue dashes show the path in data mode. SDRs in this layout are connected each other. In voice mode, SDR_1 captures audio from microphone and sends it to SDR_2 and in data mode SDR_1 reads a file from harddisk and sends it to SDR_2. Simultaneously, SDR_2 captures the incoming packets and analyze them. If the packet includes voice data than it sends it to the speaker for being played, similarly if the packet is part of a file it appends the packet to a file of which name is the ID of the sender.

In this layout, actual operating system dependent audio operations are done in AudioDevice component which is part of the core framework layer. Similarly, FileManager component implements operating system dependent file operations. The WF components use core framework components for system dependent operations. Therefore, the same waveform code can work in a different system as long as the target core framework implements the same functionality. In other words, system dependent code goes into the core framework to make waveforms more portable.

The packets travelling over the network has two sections: metadata and data. The metadata is the header part and it contains some information about the content of the packet. Figure 4.5 shows the structure of the packets between waveform and RFDevice.

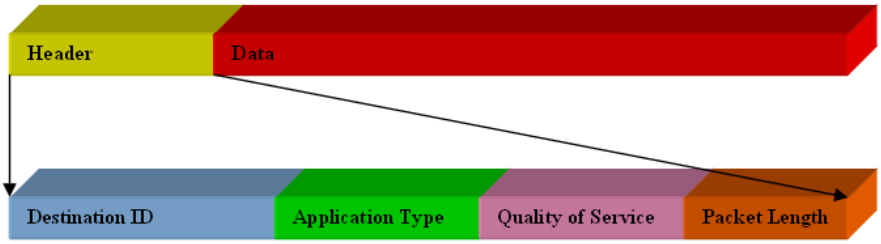

Figure 4.5. Packet Structure Between Waveform and RFDevice

The packets between waveform and the RFDevice contain a header part and a data part. The data part contains the actual data which can be voice or file data. The size of the data part is 8 kilo bytes. The header part is appended to data part and contains information about the packet. The header contains the following fields:

- **Destination ID:** Destination ID is the mobile identification number of the mobile terminal that is wanted to be communicated such as 00905351234566. The length of destination ID is 14 bytes which is the length of mobile phone numbers with international area codes.
- **Application Type:** It is a 1 byte value that can be 0 or 1. 0 means the packet contains voice data and 1 means it contains file data.
- **Quality of Service (QoS):** Quality of Service parameter can take values between 1 and 5. 1 means the data is not critical so that the mobile terminal can send it over slower base stations and it also means that the errors when transferring the data are not too important. On the other side, 5 means the data is too critical and it should be send over the fastest networks with no error even if the communication cost is higher. There is a tradeoff between the QoS parameter and the communication costs. 3 is the optimum value for QoS parameter.
- **Packet Length:** PacketLenght is a 4 byte value that is containing the length of data part of the packet. In our implementation it can take values at most 8 kilo bytes which make 8192 bytes. For files smaller than 8 kB, this parameter shows the size of the file, and also since the waveform sends data packet by packet, the last packet of a file may have a size smaller than 8 Kb, in that case PacketLength parameter shows the size of last packet. For voice packets, the PacketLength is 8 Kb by default.

The size of the fields of the header is larger than it requires. This is because of debugging purposes. For example, ApplicationType field can be 0 or 1 which can be represented by 1 bit, however since we develop a proof-of-concept application and the value of 1 bit is more difficult to debug, we have chosen to represent it with one byte. Since we focus on the design of the architecture of the application, performance issues are not too critical.

After the packets are processed and packetized by the waveform, they are sent to RFDevice by the last component of the waveform which is SendReceiveData. We have implemented RFDevice component as an RF device simulator, however it sends and receives packets by TCP/IP connection over ethernet among two nodes. RFDevice has the following main tasks:

- Talk to the base stations and select the best channel.
- Determine the communication parameters of the radio.
- Add a second header which includes additional information to the outgoing packets. The header added by RFDevice component includes the following fields:
  - **SelfID:** This is the unique identifier of the sender device. This parameter is very important for the receiver mobile terminal to understand "Where the packets are coming from?" Since SelfID does not change according to waveform, this parameter is added in RFDevice instead of waveform components.
  - **Frequency:** This parameter adds the frequency information to the outgoing packets.
  - **Coding:** Shows the type of coding.
  - **Power:** This parameters indicates the RF Power. RF power is determined according to the distance of base stations.
  - **Modulation:** This shows the modulation type of the communication.
  - **Time:** This parameter is added to add a time stamp to the packets.
  - **PacketType:** This is an additional information to distinguish packets.
  - Frequency
  - Modulation
  - Power
  - Coding

We add RF parameters to the outgoing packets. It is definite that, in a real RF communication, there would be no such parameters, and these parameters will actually be applied to the propagating signal. However in this thesis, we do not have real signals and we only have ethernet connection. What we are trying to do is to simulate RF communication over TCP/IP connection. Therefore, in order the receiving nodes to

understand the RF parameters we add these information to the packets as a second header.

4.3. Waveform Implementation

In this section, we present the implementation details by summarizing development stages[102]. We provide information about the tools we have used to design and implement our waveform and mention about the middleware and configuration management technologies in our implementation. The development strategies we present in this section do not only reflect our methodology, but also provides a guideline for the developers who intend to develop waveform applications.

4.3.1. Development Stages

When we refer waveform we mean a set files. A waveform application basically includes two types of file:

- **Binary files:** These are the actual executable files doing the job of the waveform. A waveform consists of at least one binary file, but usually it includes more than one. Each executable file has a target platform that it can run such as Linux and x86 platform.
- **XML files:** These files are the descriptor files of the binary files. Binary files cannot be used without its XML descriptors since a binary file does not provide any information such as the platform the executable is build for, external configuration parameters, CORBA interfaces it supports and so on.

We can divide waveform development into three main stages. They are:

- Component based modeling of the waveform.
- Implementing the components.
- Generating the XML configuration files.

4.3.1.1. Component Based Modeling of the Waveform. As we have mentioned early, we divide the waveform into several components. Each component can be considered as a black box which focuses at its job and talks to other components by their CORBA interfaces.

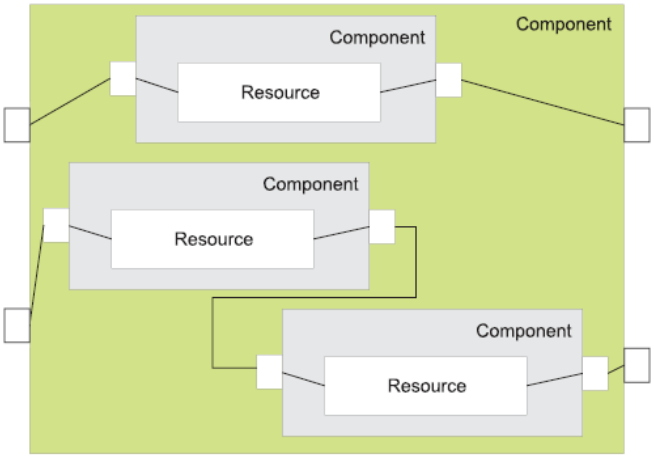

Figure 4.6. Component Based Modeling

As shown in figure 4.6, component based modeling [103] allows a very flexible architectural design. Flexible nature of component allows creation of even the waveform of waveforms.

There are several design tools for component based modeling of waveforms. Most of them are commercial products. We use Zeligsoft Component Enabler [104] design tool in this project. It is also a proprietary software. It allows visual designing of components. It also allows defining connections, dependencies and configuration parameters of waveform components.

4.3.1.2. Implementing the Components. Implementation means the realization of the design in a programming language according to the target platform. In most of the software development life cycles, the developer notices the weaknesses of the initial design when implementing the waveform. Therefore, it is usually impossible to design a system from scratch at first try. It is an evolving and repetitive process. Implementing waveform components is not an exception to this. Once a design is implemented, it usually requires redesign to maturate. If all the requirements are captured and modeled in the initial design, the following design modifications become smaller, however in worst case the developer may have to throw away the initial design and restart from the beginning if the initial design is too poor.

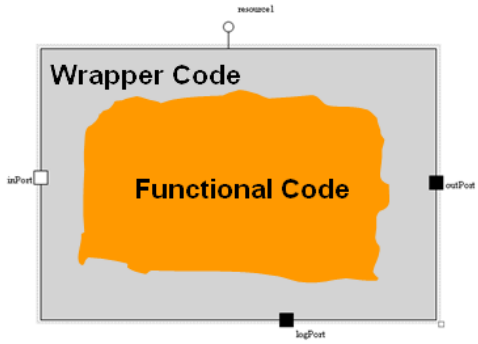

Figure 4.7. Component Structure

Figure 4.6 shows the internal structure of a waveform component. Implementation of a waveform component can be divided into two categories:

- **Implementing the functional code:** Functional code is the actual mission of the component. It includes the logic and algorithms. If we consider SDR applications, they may include a lot signal processing which consumes too much processing power. In order to handle performance issues, the functional code may be implemented in DSPs and FPGAs instead of GPP components that have less processing power. The development tool to implement functional code is dependent of the target platform. In this thesis, since our target is a PC platform (Linux based Pardus operating system and x86 based Intel processor), we implement our components in C++ programming language. We use *g++* as our compiler. There are several editors for GPP targets. We have preferred to

use *Eclipse* platform. Eclipse provides a Java based platform independent and open source development environment which can be extended by installing new plug-ins. We have installed C++ Development Tool (CDT) plug-in to be able to integrate g++ compiler with Eclipse. We have also installed *astyle* plug-in for source formatting issues.

- **Implementing the wrapper code:** The functional code itself is a highly platform dependent and cumbersome code. The wrapper code wraps the functional code with CORBA interfaces and allows it to be platform independent by the nature of CORBA. Zeligsoft Component Enabler tool that we have used in this project allows creating wrapper code by compiling the CORBA interfaces by the IDL compiler and generating some helper functions to ease development.

The key bottleneck in development stage is the lack of existence of ORB middleware for embedded targets. There are some commercial companies which are developing ORB software for FPGAs and DSP platforms. It means a functional code running on FPGA, DSP or GPP can communicate each other transparently. The problem is they usually support a sub set of the CORBA specification which limits the borders of development. We need some time to achieve true location transparency among heterogeneous processors.

4.3.1.3. Generating the XML Configuration Files. At the last step, we create the XML configuration files for waveform components. Each waveform component has at most three types of XML files. These files can be explained as follows:

- **Software Packet Descriptor(SPD):** Describes implementation details of the component.
- **Software Component Descriptor(SCD):** Describes CORBA interfaces of the component.
- **Property Descriptor(PRF):** Describes the configuration parameters of the component.

There is also a Software Assembly Descriptor (SAD) file for each waveform to describe deployment characteristics and connectivity of application components. Core framework initially reads SAD file when installing the waveform. SAD file includes paths to other configuration files for each component so that the core framework can locate them.

Preparing these XML files manually is too difficult and error-prone process. In this thesis, we have used Zeligsoft Component Enabler to create these files according to our waveform model, however manual modification of these files is sometimes required in case of errors when running the waveform. In addition, it may be necessary to change initial configuration parameters defined in XML files with a text editor.

4.4. Applying Design Patterns

In this section, we propose some design patterns for our SDR layers and we provide possible application areas for them [105]. We present the application of Factory Method, Chain of Responsibility, Adapter, State, Singleton, and Facade patterns and provide some UML diagrams to illustrate our ideas.

4.4.1. Factory Method

Factory Method pattern defines an interface for creating an object, but let subclasses decide which class to instantiate. It lets a class defer instantiation to subclasses.

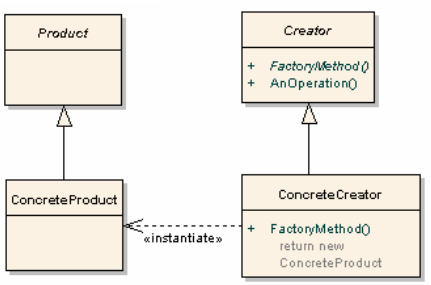

Figure 4.8. Factory Method Pattern

As Figure 4.8 shows, the pattern uses two types of classes. The Product classes, which are the classes that make up the application and the Creator classes, which are responsible for defining the Factory methods used to create instances of Product objects. The Creator class defines the factory method, which returns an object of type Product. The Concrete classes provide the appropriate implementation for their respective base class.

Factory method is very useful and common design pattern to solve portability problems. It can be used to separate Operating System and Object Request Broker specific implementations with the rest of the system. Factory pattern uses a factory which decides which specific subclass to handle the request of the client.

This pattern is also suitable to manage configuration specific issues. Factory class may be used to check the configuration value when returning the possible concrete handler class. It is obvious that changing the configuration parameter will make the factory to return a new appropriate concrete handler. Also the factory class may keep a list of the created objects and can be used to kill or modify them by only traversing the managed list of instances.

Figure 4.9-a shows an example scenario where Factory Method pattern is used to separate OS specific codes from the rest of the system. In this diagram, Process is the interface class that defines mandatory functions that every derived class should implement. ProcessLinux and ProcessVxWorks classes concretely defines OS specific execute and terminate functions and overrides generic Process interface. In this scenario, ProcessFactory checks the OS configuration parameter of the core framework and returns appropriate Process class. Regardless of the returning concrete Process class, the client can call execute and terminate operations on the returned object.

Figure 4.9-b shows an example usage of the Factory pattern to decouple the applications from ORB specific functions. ORBFactory class has a getORBLibrary method which checks the configuration parameter of the core framework and returns the concrete ORBLibrary class so that the clients that need CORBA functions can call

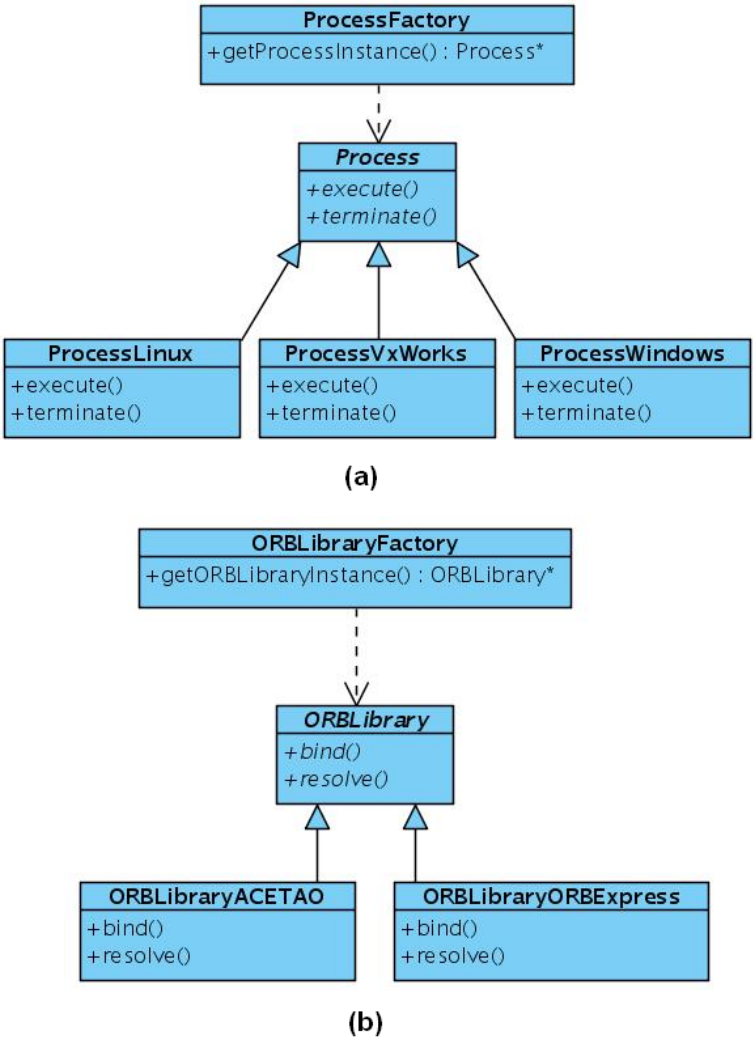

Figure 4.9. Example Usage of Factory Method Pattern

ORB functions independently.

It is worth noting that Factory Method pattern is very helpful to deal with possible future changes. Changing the configuration parameter of the factory makes the system to behave according to the new situation without affecting the existing codes. This also allows the system to extend by defining new derived concrete classes.

4.4.2. Chain of Responsibility

Chain of Responsibility pattern is used to avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. It chains the receiving objects and passes the request along the chain until an object handles it.

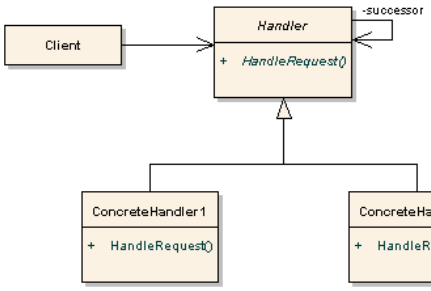

Figure 4.10. Chain of Responsibility Pattern

This pattern can be applied to many cases while developing software for SDR systems. SCA standard tells developers to design their system in a hierarchical manner which means dividing the software architecture into collaborating components. Such a distributed system requires a strong management mechanism for the responsibilities of the components. From that point of view, this design pattern can be considered to be a useful blue print to handle responsibility related issues.

SCA interfaces defines port concept as a communication mechanism between components. A port represents a CORBA interface of which reference can be transferred between components so that distributed components can make CORBA calls on each other. Efficient use of port mechanism in conjunction with Chain of Responsibility pattern can let developers to manage object responsibilities even the objects are distributed.

Figure 4.11 illustrates a generic usage scenario which can be applied to similar situations. In this figure, four components of a waveform are shown. They are all connected to each other by using port mechanism. In this layout, DataSource component is responsible to get some data to be processed by the waveform and the other three

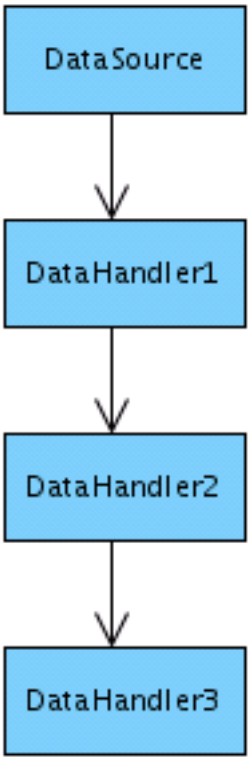

Figure 4.11. Example Usage of Chain of Responsibility Pattern

components are chained each other and each of the component implements a different algorithm to process the data. In this example scenario, DataSource component is not aware of which component in the chain will handle the request and it does not have to. It only concentrates on the job of fetching the data to push to the chain. In this scenario, each DataHandler component checks some internal or external parameters to decide whether to handle the incoming data or not. The parameters that can be checked during deciding stage can be permissions, capacity values, priorities, dependencies, performance requirements and structural properties of the incoming data or so on.

It is obvious that this pattern lets insertion of new handlers into the chain without affecting the rest of the system. It is also valid for the removal case. The developer does not have to modify any component for any changes. This behavior can be very important for the systems that requires frequent modification of the code according to changing conditions such as development phases. It can also be an interesting idea

to chain the instantiations of the same component to balance the work load among different processors.

4.4.3. Adapter

Adapter pattern is used to convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

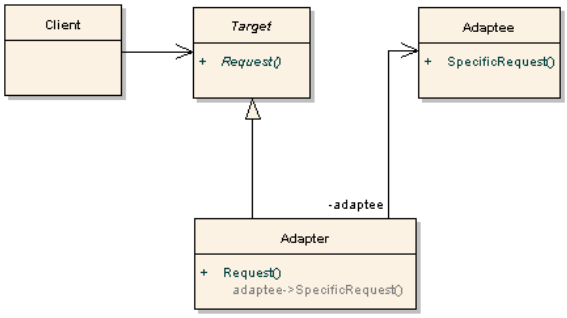

Figure 4.12. Adapter Pattern

Adapter pattern has also very common usage in SCA implementations. It allows legacy codes that do not support SCA interfaces to work together with the SCA codes. It simply adapts the old interface to the new one.

Adapter pattern is typically not used when designing a new system from scratch but rather used to port existing codes from one interface to another.

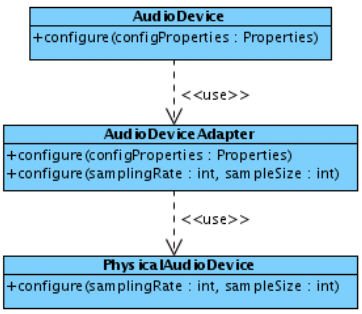

Figure 4.13. Example Usage of Adapter Pattern

Figure 4.13 shows an example usage of the Adapter pattern. In this scenario

adapter class adapts configure methods of different audio device interfaces. As shown in the figure, legacy PhysicalAudioDevice class has a configure method this accepts integer configuration parameters, whereas SCA compliant AudioDevice class accepts Properties structure as input. AudioDeviceAdapter class translates parameters of these classes between each other so that they can work together.

4.4.4. Singleton

Singleton pattern is applied to ensure a class only has one instance, and provide a global point of access to it. It is a relatively simple pattern to apply.

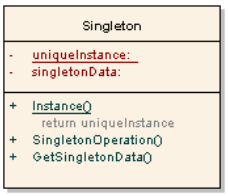

Figure 4.14. Singleton Pattern

In the programming domain, it is a very common situation that a class is required to have only one instance. For the SCA point of view, Singleton pattern can be frequently used. For example, Device classes that wraps a specific hardware usually requires having only one instance, because the managed device usually cannot be initialized more than once and the capacity values should be under control of a single capacity manager. Another example can be the ORBLibrary classes that initialize and manage POA (Portable Object Adapter) according to a specific ORB policy. Singleton pattern can be used together with the Factory Method pattern to ensure returned concrete classes to have only one instance. In this case, Factory class can check the instance count of the singleton objects and return the same object whenever it creates an instance.

4.4.5. State

State pattern allows an object to alter its behavior when its internal state changes. The benefit of State pattern is that state specific code is localized in the class that represents that state.

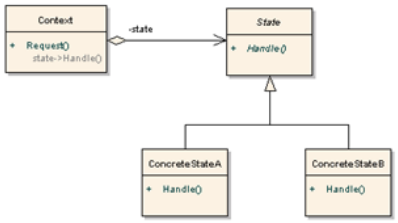

Figure 4.15. State Pattern

SCA defines three types of state types for Device classes. They are OperationalState, AdminState and UsageState. OperationalState can be ENABLED or DISABLED and indicates whether the device is functioning or not. AdminState keeps track of the permission or prohibition against using the device and it can take values of LOCKED, SHUTTINGDOWN or UNLOCKED. Finally, UsageState defines Devices usage state and can be IDLE, ACTIVE or BUSY. IDLE means that Device is not in use, BUSY corresponds to Device is in use and no capacity is left for allocation and ACTIVE shows that Device is in use and it still has some capacity for allocation. In addition to built-in states, it is possible to add user defined states for different situations of the components.

Applying State pattern helps developers to separate state dependent operations from the rest of the functional code of the components and it reduces complexity.

4.4.6. Facade

Facade pattern provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. It also simplifies and beautifies an existing cumbersome class by behaving as a door to its complex interface. It means it works as an intermediator between the client and the

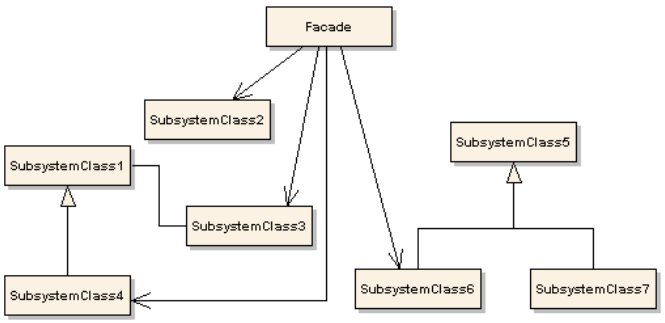

Figure 4.16. Facade Pattern

subsystem. Facade should not be the part of the subsystem, if this is the case it should move to the subsystem and a new Facade class should be generated.

From SCA point of view, Facade pattern can be applied while porting non-SCA legacy codes to SCA compliant wrapper codes. SCA wrapper codes may use Facade classes to access the legacy parts so that the developer may not spent time to re-implement already existing functional codes. In addition, Facade pattern may be applied to collect separate CORBA interfaces into a single CORBA interface.

5. EVALUATIONS

In this chapter, the evaluation results of thesis are presented. Evaluation results can be investigated in two sections. In the first section, our proposed channel selection algorithm is evaluated for different scenarios. In each scenario, different test cases are generated to better understand the performance of our algorithm.

In the second section, our SDR design and its layers is evaluated. We summarize performance statistics, and static code analysis of our waveform and discuss the lessons learned and troubles encountered while designing, implementing and running our application. We also evaluate our architecture by considering CR requirements and software engineering metrics.

5.1. Evaluation of Proposed Channel Selection Algorithms

In this section, we present our evaluation results for the proposed channel selection algorithms AHS and PCS. We compare our results with different channel selection algorithms. They are random channel selection algorithm and optimum channel selection algorithm. AHS and PCS is explained in previous chapters in more details. In random channel selection algorithm, channel is selected randomly when handoff is required. Optimum channel selection algorithm is the theoretical limit that can be achieved when selecting channel. Because it is based on selecting the channel by exactly knowing future usage timings of all channels and the user. Therefore it selects the best available channel at all times to minimize handoff.

We evaluate our results for the following criteria:

- **Handoff Average Per Call:** This is the main criterion for our evaluations. We aim to minimize average handoff per call to achieve better connection holding performance.
  $HandoffAveragePerCall = \frac{HandoffCount}{TotalSuccessfulConnectionCount}$

- **Probability of Blocking:** This criterion measures the probability of unsuccessful connection attempt. This criterion can take values between 0 and 1 where 0 means no connection is blocked and 1 means there is no successful connection.
  $Probability of Blocking = \frac{BlockCount}{TotalConnectionCount}$
- **Probability of Dropping:** This criterion indicates the probability of unsuccessful handoff. This value range between 0 and 1 where 0 means all handoff attempts are successful and 1 means no handoff is achieved.
  $Probability of Dropping = \frac{DropCount}{TotalConnectionCount}$
- **Average Drop Percentage:** Average drop percentage shows the average dropping percentage of the started connections. This criterion can take values between 0 and 100. For example, if ADP is 30, it means that connections has dropped after completing 30% of the total connection time. This criterion is helpful to understand other criteria.
  $AverageDropPercentage = \frac{AverageDroppingPercentageofDroppedConnections}{DropCount}$
- **Total Average Completion Percentage:** This criterion shows the percentage of completion for all started connections.
  $TotalAverageCompletionPercentage = \frac{AverageCompletionPercentageofAllConnections}{TotalConnectionCount}$

5.1.1. Evaluation Tools

We have implemented a simulator tool and an input generator for our evaluation purposes. Input generator tool is used to generate different test cases for the simulator tool according to the given set of parameters. It can generate scenarios for the different combinations of the following parameters:

- Initial user holding time average
- User holding time average increase at each step
- Initial user blank time average
- User blank time average increase at each step
- Initial channel holding time average
- Channel holding time average increase at each step
- Initial channel blank time average

- Channel blank time average increase at each step
- Secondary user percentage in channels
- Secondary user percentage increase at each step
- Test set count
- Maximum time for the simulation
- Number of runs of each test set
- Initial number of channels
- Increase of the number of channels at each step
- Error rate for the generated test sets

5.1.2. Evaluation Scenarios

We have generated different scenarios with our simulator input generator tool to evaluate the performance of our proposed channel selection algorithm. We have evaluated only one parameter at each scenario and fixed all other parameters during these tests.

5.1.2.1. Evaluation of Channel Count. Channel count parameter has an important effect on the behavior of the channel selection algorithms. In this test case, this effect is investigated. We have generated a scenario, where the number of selectable channels are increased at each step. In the starting condition, there exists only one channel, whereas on the last step there are 100 channels. In these tests, all other parameters are fixed in order to visualize the channel count effect only. Table 5.1 summarizes our scenario.

Figure 5.1 illustrates our test results. In the starting condition, where there is only one channel, it noticed that there is no handoff. This is a result of the fact that user connection requests can not take start because of the lack of available channels. In addition, since there is only one channel, most of the started connections drop in case of a primary user begins transmitting. However, having only one channel is an extreme case and it is far from the reality where CR operates.

Table 5.1. Evaluation of Channel Count Scenario Parameters

Test Parameter	Value
Initial user holding time average	400
User holding time average increase at each step	0
Initial user blank time average	25
User blank time average increase at each step	0
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	500
Channel blank time average increase at each step	0
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	100
Maximum time for the simulation	360000
Number of runs of each test set	10
Initial number of channels	1
Increase of the number of channels at each step	1
Error rate for the generated test sets	5

As the number of selectable channels reaches around 6 to 10, handoff count begins increasing. This is a result of the fact that having more channels prevents blocking or dropping by allowing the CR jumping to alternative channels which means more handoffs.

The graph shows that increasing channel count more than 10, generates a slowly decreasing but mostly horizontal pattern. This pattern shows that limited number of channels does not allow efficient operation for channel selection algorithms. An outcome of this situation is that, as the number of selectable channels increase, importance of the selected algorithm also increases. As the channel count reaches 50, AHS algorithm starts performing better than other algorithms and starts converging to optimum handoff average.

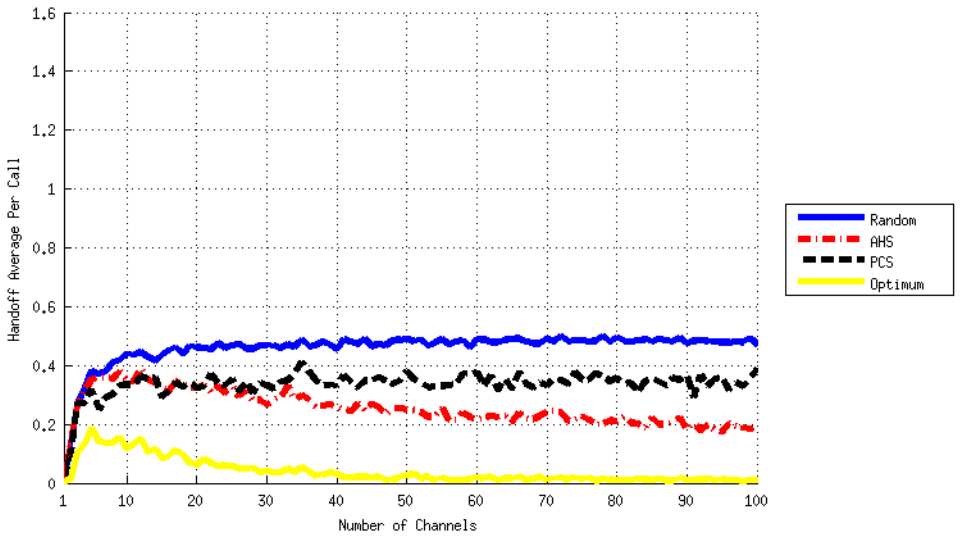

Figure 5.1. Evaluation of Channel Count

Another result of this test is that selecting channel count more than 10, where graph becomes horizontal, is necessary in other evaluation scenarios in order to let them become independent of the channel count parameter and understand the effect of the tested parameters better.

5.1.2.2. Evaluation of Secondary User Probability. In this scenario, the effect of secondary user probability in channels is evaluated. Secondary user probability is the probability of a channel user to become secondary user. For example, if secondary user probability is 25%, it means that 25% of the users in a channel are secondary user and %75 of the users are primary user. In the starting condition, secondary user probability is 0 which means that all users in channels are primary user. Similarly, in the last test set, secondary user probability becomes 1 which means that all users in channels are secondary user. At each test set, probability of the secondary users is increased by 10%. In each test set, all other parameters are fixed including channel usage patterns and user connection requests in order to see the effect of secondary user probability only. Table 5.2 summarizes our scenario.

Table 5.2. Evaluation of Secondary User Probability Scenario Parameters

Test Parameter	Value
Initial user holding time average	400
User holding time average increase at each step	0
Initial user blank time average	25
User blank time average increase at each step	0
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	500
Channel blank time average increase at each step	0
Secondary user percentage in channels	0
Secondary user percentage increase at each step	10
Test set count	11
Maximum time for the simulation	360000
Number of runs of each test set	5
Initial number of channels	50
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

Figure 5.2 presents our simulation results.

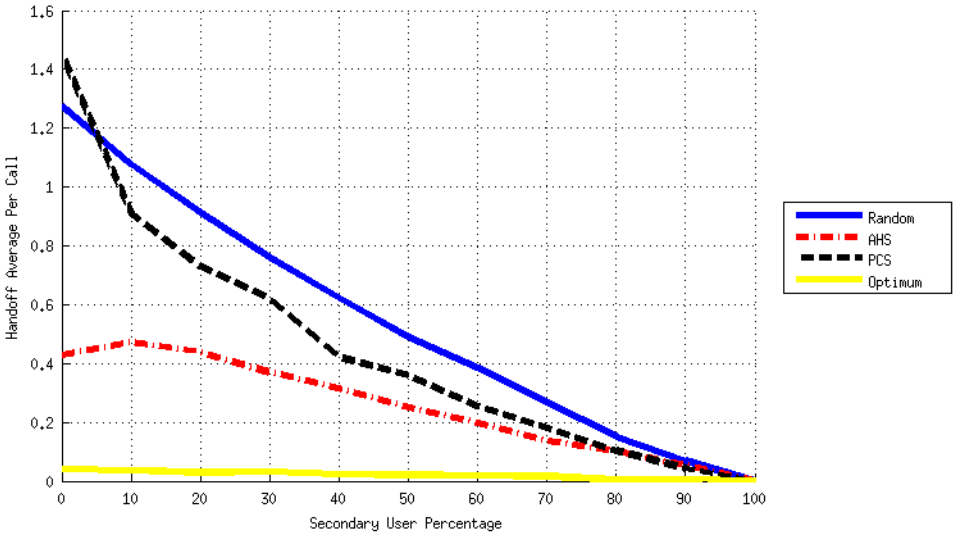

Figure 5.2. Evaluation of Secondary User Probability

This test shows that increasing secondary user probability in channels, affects handoff average directly. As the number of secondary users increases, handoff average decreases respectively.

If all users are secondary user, than it is clear that there is no need for handoff. This is a result of the fact that the channel selection procedure where our CR user, which is also a secondary user, has precedence of using the selected channel over other secondary users as long as it selects the channel first. In other words, if our CR user is using a channel, it can use it unless a primary user is detected.

Average handoff performance of AHS converges to the optimum handoff average as the number of secondary users increases. In the first test case where all users are primary, AHS achieves better handoff performance compared to PCS and random channel selection algorithm. But the difference between algorithms decreases as the number of secondary users approaches to 100%.

5.1.2.3. Evaluation of User Holding Time. In this scenario, we evaluate the effect of the user holding time. We run channel selection algorithms for different user holding time averages. In each step, we increase only the holding time average of the user where all other parameters are fixed. In the starting condition, the user holding time average is 20 and it is increased by 20 in each step. Table 5.3 summarizes our scenario.

Table 5.3. Evaluation of User Holding Time Scenario Parameters

Test Parameter	Value
Initial user holding time average	20
User holding time average increase at each step	20
Initial user blank time average	25
User blank time average increase at each step	0
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	500
Channel blank time average increase at each step	0
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	40
Maximum time for the simulation	360000
Number of runs of each test set	10
Initial number of channels	50
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

Figure 5.3 illustrates our simulation results. The handoff average presents an ascending pattern as the user holding time average increases. This is a result of the fact that longer user holding time requires more handoffs as long as other conditions including the channel usage pattern are fixed.

An outcome of this test is that, although the holding time average of the user is increased by 20 at each step, handoff average increasing rate differs during the graph. There exists a break point when user holding time average is around 500 which is

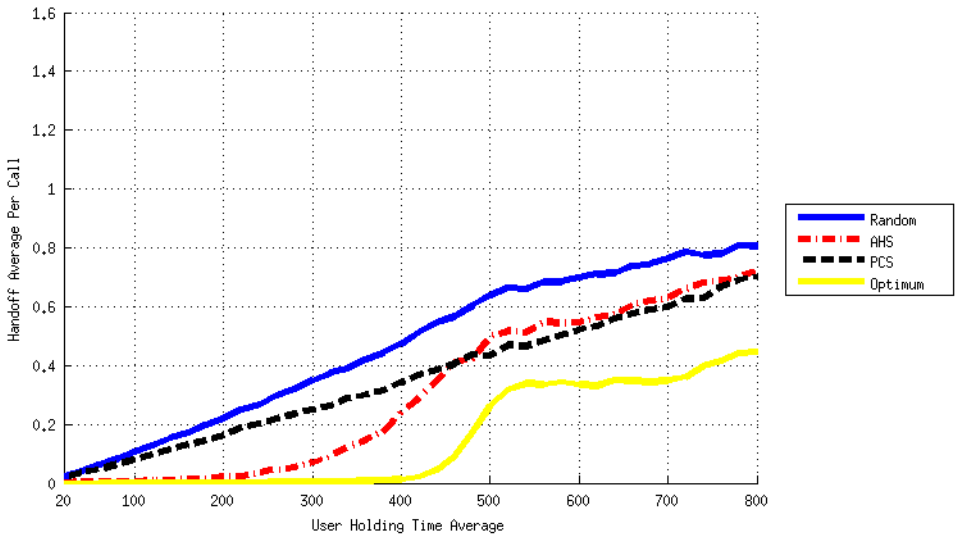

Figure 5.3. Evaluation of User Holding Time

equal to the average channel blank time. This is a result of the fact that when the user holding time is less than channel blank time, user connections can fit into the gaps of the channels which prevents handoffs. When the user holding time is around channel blank time, handoff average shows a sharply ascending pattern. Increasing holding time values more than channel blank time makes the graph increase with a lower slope. This behavior shows that handoff average of the CR user is directly proportional with the holding time average, however the handoff average of the user is strictly dependent of the user holding time.

5.1.2.4. Evaluation of User Blank Time Between Calls. In this scenario, the effect of user blank time between calls is investigated. We have generated test sets where user blank time between calls is increased by 25 in each step. In the starting condition user blank time is set to 25 and at the end of the 160 steps it becomes 4000. Table 5.4 summarizes our scenario.

Table 5.4. Evaluation of User Blank Time Between Calls Scenario Parameters

Test Parameter	Value
Initial user holding time average	400
User holding time average increase at each step	0
Initial user blank time average	25
User blank time average increase at each step	25
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	500
Channel blank time average increase at each step	0
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	160
Maximum time for the simulation	360000
Number of runs of each test set	5
Initial number of channels	50
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

As shown in figure 5.4, increasing user blank time between calls does not change handoff average significantly. The reason is that the handoff average is calculated as the rate of handoff count to total successful connection number. Increasing the gaps between user calls decreases both the total connection number and the total handoff count, so that the rate does not change significantly. Therefore, handoff probability graph illustrates a horizontal pattern until the end of the simulation.

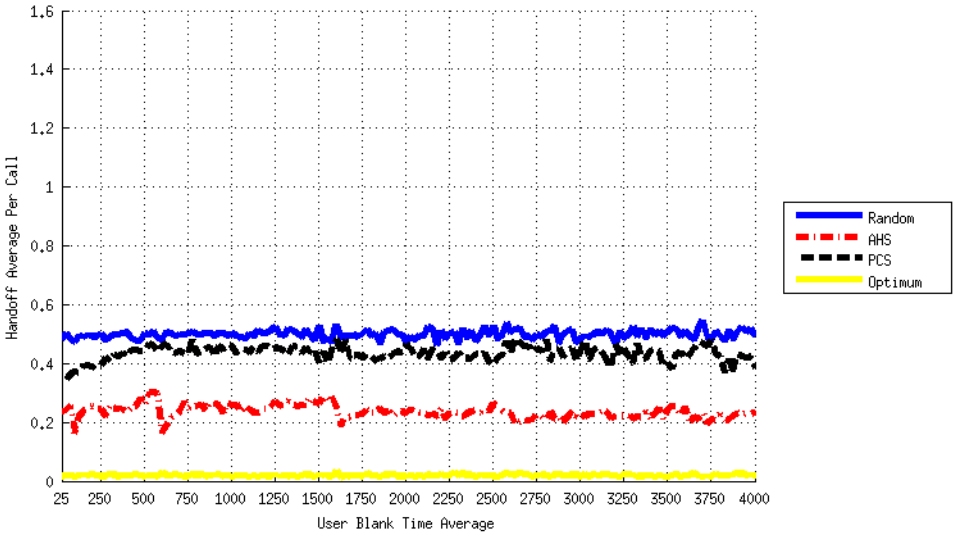

Figure 5.4. Evaluation of User Blank Time Between Calls

5.1.2.5. Evaluation of Channel Holding Time. In this scenario, the effect of channel holding time is evaluated. We have generated 160 test set where channel holding time is started from 20 and increased 20 at each step so that at the final step it becomes 4000. Table 5.5 summarizes our test parameters for this scenario.

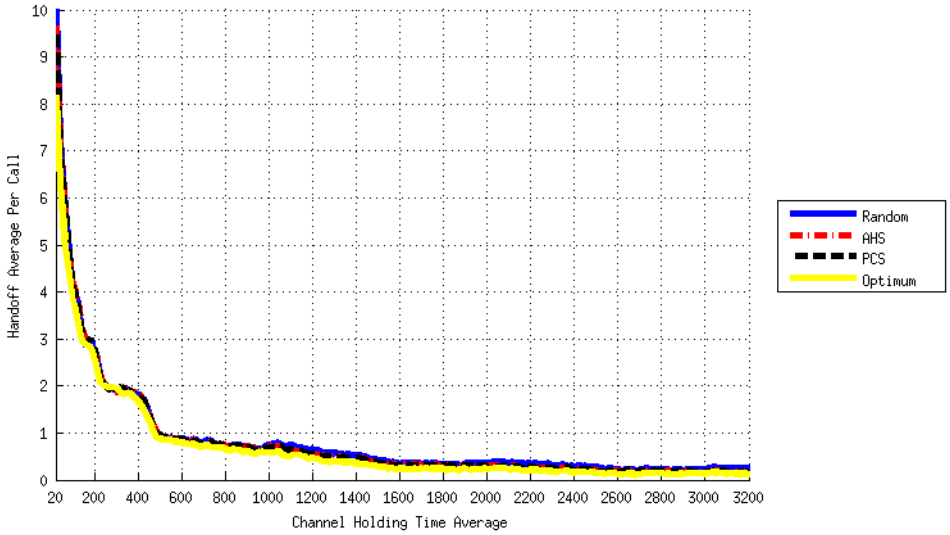

Figure 5.5. Evaluation of Channel Holding Time

Figure 5.5 shows our simulation results for this test. At first glance, increasing channel holding time is expected to increase handoff average. However, the graph is surprising. Understanding the graph requires more detailed analysis. At the beginning

Table 5.5. Evaluation of Channel Holding Time Scenario Parameters

Test Parameter	Value
Initial user holding time average	500
User holding time average increase at each step	0
Initial user blank time average	500
User blank time average increase at each step	0
Initial channel holding time average	20
Channel holding time average increase at each step	20
Initial channel blank time average	20
Channel blank time average increase at each step	0
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	160
Maximum time for the simulation	360000
Number of runs of each test set	10
Initial number of channels	100
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

step of the simulation user holding time average is 500, channel holding time average is 20, and channel blank time average is 25. This means connection time average of the user is 20 times longer than the blank time average of the channels. Therefore, handoff count of the user becomes maximum at the first step. In the next steps, only the channel holding time average increases, so that user makes less handoff to complete for the same amount of connection requests. As a result, handoff average graph illustrates a decreasing pattern.

5.1.2.6. Evaluation of Channel Blank Time Between Calls. In this scenario, channel blank time between calls is evaluated. We have generated 160 test sets where channel blank time is started from 20 and increased 20 at each step so that at the final step it becomes 3200. Table 5.6 summarizes our test parameters for this scenario.

Table 5.6. Evaluation of Channel Blank Time Between Calls Scenario Parameters

Test Parameter	Value
Initial user holding time average	500
User holding time average increase at each step	0
Initial user blank time average	500
User blank time average increase at each step	0
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	20
Channel blank time average increase at each step	20
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	160
Maximum time for the simulation	360000
Number of runs of each test set	10
Initial number of channels	100
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

Figure 5.6 shows our simulation results for this test. It shows that increasing channel blank time decreases handoff average as other parameters are fixed. This is a result of the fact that increasing the gaps between channel connections allows the user to complete his connection without jumping to other channels. Since the handoff average is calculated as the rate of total handoff count to total successful connection number and increasing channel blank time decreases total handoff count for the same amount of connection, we observe a descending handoff average graph.

Another outcome of this scenario is that AHS makes a break point at test set 25

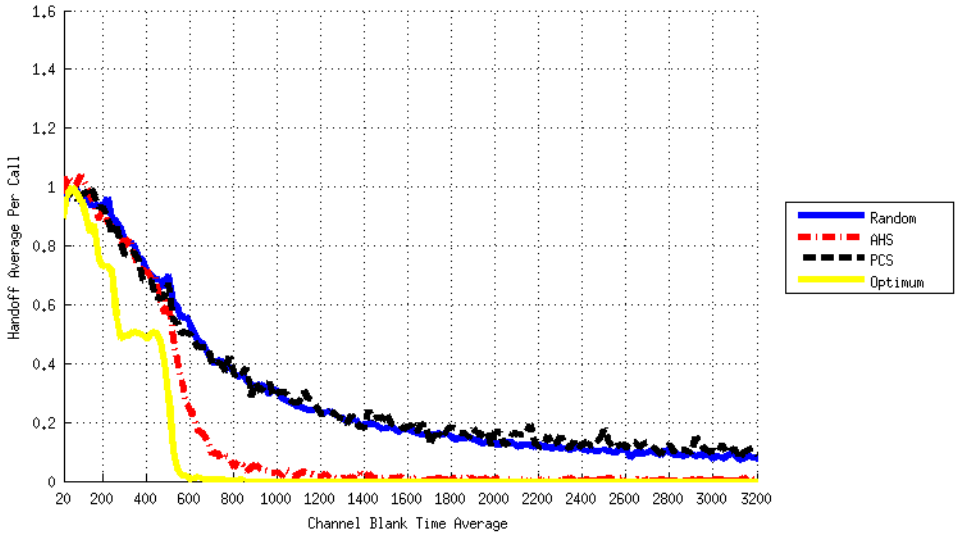

Figure 5.6. Evaluation of Channel Blank Time Between Calls

where channel blank time becomes 500 which is equal to the user holding time average. At that point AHS starts to converge to the optimum handoff average. It is noticed that before that point all of the algorithms generates similar results. After test set 50 where channel blank time becomes 1000, AHS generates results that are very close to optimum. The graph shows that as the channel blank time is continued to be increased all of the algorithms converges. This is a result of the fact that user starts to complete his connections without handoff after channel blank time is larger enough.

5.1.2.7. Evaluation of Channel Sensing Error Rate. In this scenario, we evaluate the effect of sensing error rate. We have generated a scenario where user holding time is increased by 20 at each step and becomes 800 at the end of 40 steps. We have repeated the same scenario for different sensing error rates. In this test, if sensing error rate is 0, it means that our CR user is capable of detecting the channel user types (primary user or secondary user) perfectly. Similarly, sensing error rate 100 is the worst case where channel user types are never detected correctly. In this test, if the channel user type can not be detected correctly, interference occurs and connections fail. Therefore, in case of sensing error, connections of the CR user and selected channel user are cancelled at run-time during simulation. Table 5.7 summarizes our test parameters for this scenario.

Table 5.7. Evaluation of Channel Sensing Error Rate Scenario Parameters

Test Parameter	Value
Initial user holding time average	20
User holding time average increase at each step	20
Initial user blank time average	25
User blank time average increase at each step	0
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	500
Channel blank time average increase at each step	0
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	40
Maximum time for the simulation	360000
Number of runs of each test set	10
Initial number of channels	50
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

Figure 5.7, 5.8, and 5.9 presents our simulation results.

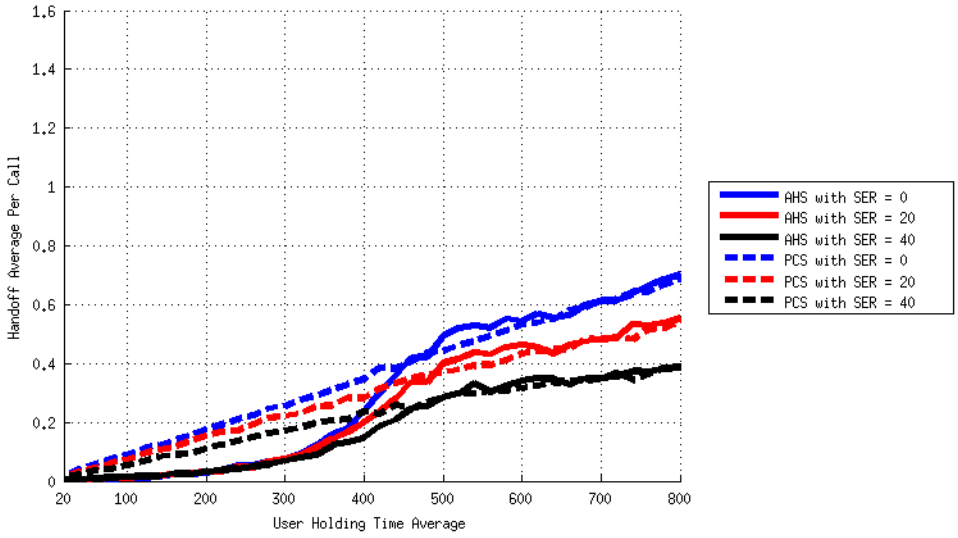

Figure 5.7. Evaluation of Sensing Error Rate for Handoff

Handoff average graph shows that increasing sensing error rate decreases the handoff average increasing rate. This is a result of the fact that increasing sensing error rate also increases dropping probability and blocking probability. For example, in our experiments we have observed that when sensing error rate is 100, blocking probability becomes 1 for all test sets which means all started connections are blocked. Therefore handoff average becomes 0 when sensing error rate is 100. It is noted that AHS algorithm performs better than PCS algorithm as the user holding time average increases. However, the difference between the algorithms decreases as the sensing error rate increase.

Probability of blocking graph shows that blocking probability is directly proportional with sensing error rate. Increasing sensing error rate from 0 to 100 has also increased blocking probability at the same rate. This is a result of the fact that user connection is blocked when sensing error occurs at the beginning of user connection request.

Another outcome of this test is that probability of blocking is not dependent on user holding time. Although the user holding time is increased at each step we observe that probability of blocking does not change. This is a result of the fact that blocking is

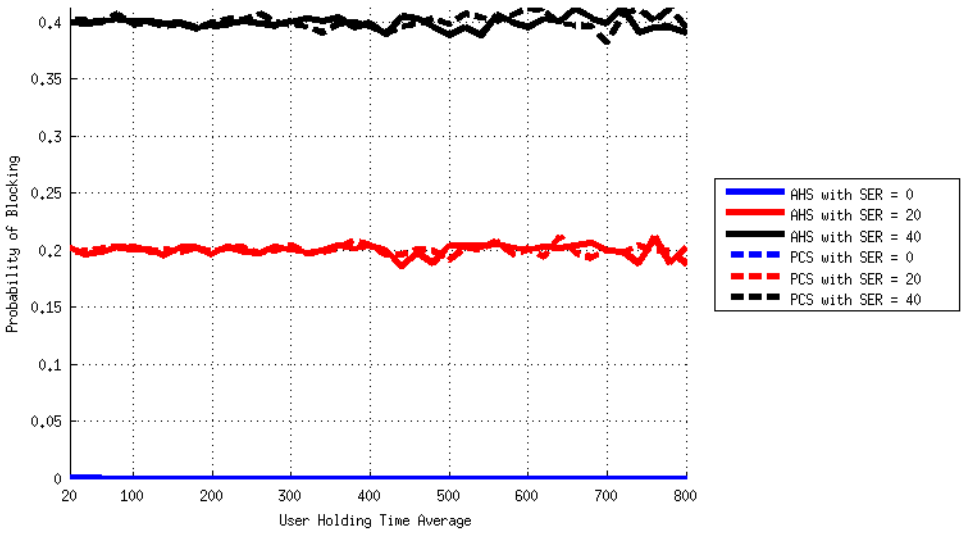

Figure 5.8. Evaluation of Sensing Error Rate for Blocking

determined at the beginning of user connection request, therefore it is not related with the holding time of the user. Also, it is observed that AHS and PCS algorithms does not change blocking probability significantly as the user holding time average increases.

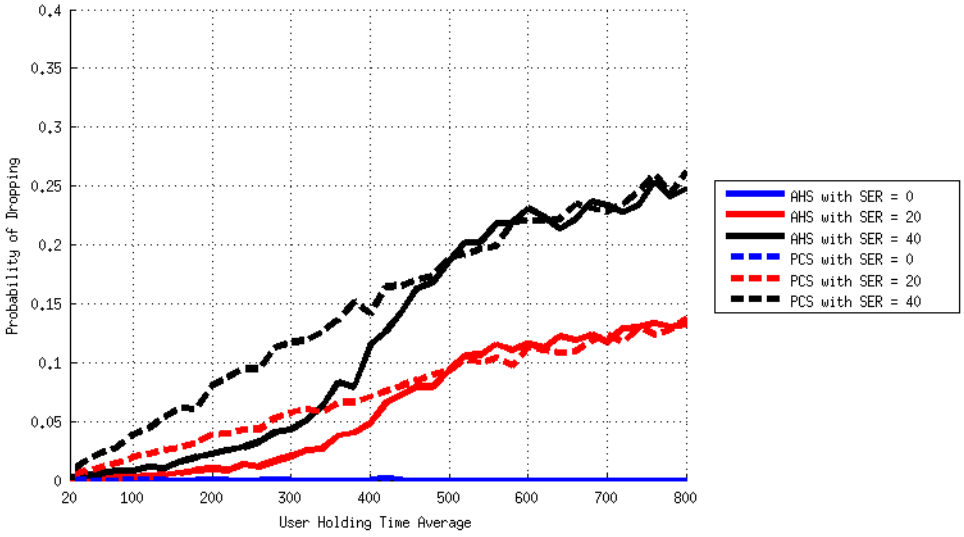

Figure 5.9. Evaluation of Sensing Error Rate for Dropping

Probability of dropping graph shows that as the sensing error rate increases, dropping probability also increases. This is a result of the fact that user connection is dropped in case of sensing error while jumping to other available channels. Therefore, increasing sensing error also increases dropping probability.

5.1.2.8. Evaluation of Alpha Values. In AHS, channel is selected according to the average holding time of the user. In this scenario, we multiply average holding time of the user with an alpha constant when selecting channel. Therefore, channel can be selected according to different user average holding time values. In this scenario, effect of different alpha constants is evaluated. We have generated a scenario where user holding time average is started from 20 and increased 20 at each step and becomes 3200 at the final step. In this scenario all other parameters are fixed. We run AHS algorithm for different alpha values. In the initial case, we select alpha as 0.25 and increase it by 0.25 in each run until it becomes 2.0 at the final case. For alpha values less than 1.0, AHS selects channel according to smaller average holding time values of the user. Similarly for alpha values greater than 1.0 AHS selects channel according to greater average holding time values of the user. Table 5.8 summarizes our test parameters for this scenario.

Table 5.8. Evaluation of Alpha Values Scenario Parameters

Test Parameter	Value
Initial user holding time average	20
User holding time average increase at each step	20
Initial user blank time average	25
User blank time average increase at each step	0
Initial channel holding time average	500
Channel holding time average increase at each step	0
Initial channel blank time average	500
Channel blank time average increase at each step	0
Secondary user percentage in channels	50
Secondary user percentage increase at each step	0
Test set count	160
Maximum time for the simulation	360000
Number of runs of each test set	10
Initial number of channels	50
Increase of the number of channels at each step	0
Error rate for the generated test sets	5

Figure 5.10 shows our simulation results. Graph shows that alpha values greater than 1 shifts the handoff average graph to the left side. In addition, alpha values less than 1 does not affect graph significantly and generates a similar pattern as alpha 1 does. Shifting handoff average graph to the left means that more handoff occurs for smaller user holding time values. For example, at test set 15 where user holding time is 300, selecting alpha as 2 yields 0.4 handoff per call where selecting alpha 1.0 yields 0.2 handoff per call. It can be concluded that selecting alpha values larger than 1, which means searching for larger channel gaps, increases handoff average. In addition, selecting alpha values smaller than 1, which means selecting smaller channel gaps, does not have significant affect on handoff average.

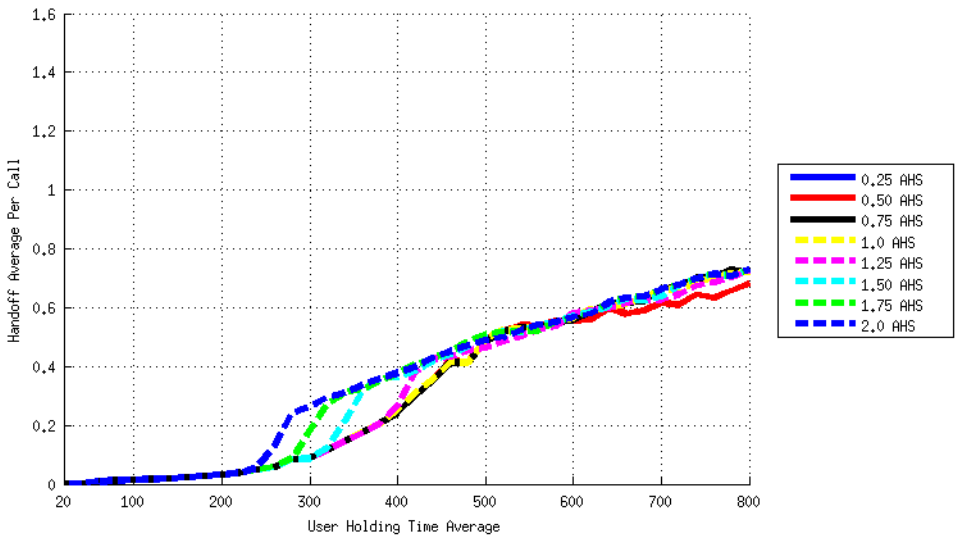

Figure 5.10. Evaluation of Alpha Values

5.2. SDR Evaluation

In this section, we present some tables and figures that reflect our evaluation results.

5.2.1. Static Code Analysis

Table 5.9 shows the files of the implemented waveform and their properties.

Table 5.9. Waveform File Analysis

Component	Number of Files	Size (kB)
Source file (cpp)	37	461
Header file (h)	53	225
Binary file	8	17844
XML	22	41
IDL	3	55
DTD	8	15
Zeligsoft model	1	310
Waveform (All files)	38	17485
Waveform (All files in jar)	1	3584

As shown in the table our waveform is implemented in C++ programming language. It is composed of 38 files. These files contain executables and configuration files for the waveform components. SCARI-Open core framework only installs the waveforms in the form of a jar file. The jar extension here, should not be mixed with java executables. It is nothing more than a zip file with a jar extension. We have generated the jar file by compressing 38 waveform component to a zip file and changing the extension to jar. It is advantageous to follow this methodology because of the following reasons:

- The total size of the waveform is reduced which is useful to distribute it.
- Jar file acts as a container which keeps every thing together.

- Number of waveforms that can be installed to limited memory devices is increased by reducing application size.

Table 5.10 shows the properties of each waveform component in more details.

Table 5.10. Waveform Component Analysis

Component	Binary Size (kB)	# XML files	# CORBA Connections
ReadData	2158	3	2
CaptureData	2181	3	3
WriteData	2175	2	2
PlayData	2180	2	3
ProcessTx	2190	3	4
ProcessRx	2190	3	4
SendReceiveData	2195	3	3
AssemblyController	2158	3	8

The table shows the binary size of each waveform executable, number of configuration files and required CORBA connections for each component. AssemblyController is the manager component of the waveform, therefore it has the most number of CORBA connections to other components. In addition, size of binaries does not differ significantly, this is because most of the binary code is generated from wrapper code as shown in table 5.11.

Wrapper code is the code to adapt the component to the core framework. It is the implementation of CORBA interfaces and it is usually automatically generated by using some tools. Functional code is the actual code implemented by the developer and it makes the job of the component. Table 5.11 presents some comparison of these codes in our implementation.

Table 5.11. Waveform Component Analysis

Component	FnC	WrC	ToC	FoT	WoT	FLOC	WLOC	TLOC
ReadData	27985	51745	79730	35.10	64.90	933	1725	2658
CaptureData	28044	52554	80598	34.79	65.21	935	1752	2687
WriteData	11983	50552	62535	19.16	80.84	399	1685	2085
PlayData	9853	50826	60679	16.24	83.76	328	1694	2023
ProcessTx	12483	55319	67802	18.41	81.59	416	1844	2260
ProcessRx	14308	55247	69555	20.57	79.43	477	1842	2319
SendReceiveData	18241	51638	69879	26.10	73.90	608	1721	2329
AssemblyController	15876	65300	81176	19.56	80.44	529	2177	2706

The fields of the table can be explained as follows:

- **FnC:** Size of functional code in bytes.
- **WrC:** Size of wrapper code in bytes.
- **ToC:** Size of total code in bytes.
- **FoT:** Percentage of functional code.
- **WoT:** Percentage of wrapper code.
- **FnLOC:** Functional lines of code.
- **WrLOC:** Wrapper lines of code.
- **ToLOC:** Total lines of code.

5.2.2. Performance Analysis

In this section we present performance results of our implementation. The implemented waveform has been tested on a laptop with the following configuration:

Table 5.12. Test Configuration

Component	Configuration
CPU	Intel based Centrino Duo 2 GHz
RAM	2 GB
Hard disk	20 GB Linux partition on a 160 GB and 5400 RPM hard disk
Soundcard	Full duplex sound card
Ethernet	Intel PRO 100 ethernet card
Operating System	Pardus 2008
WF ORB	ACETAO
Core framework	SCARI-Open
CF ORB	Java ORB

Each scenario has been tested several times and the averages are taken. In the first scenario, we have tested timings for each life cycle operation of our PC based SDR platform. Figure 5.11 shows the life cycle operations and their precedence.

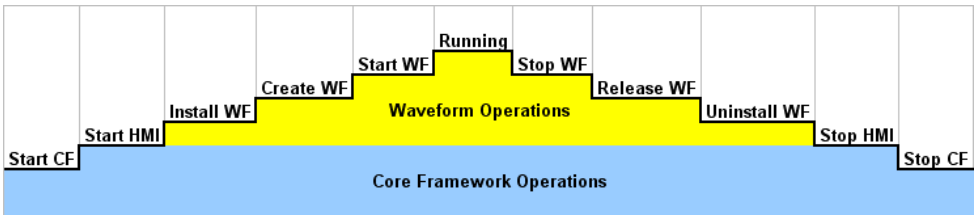

Figure 5.11. SDR Life Cycle

Life cycle of a SDR platform can be investigated in two categories:

- **Core framework operations:** It includes operations related to core framework and human machine interface (HMI) which is the user interface of the core framework. Separating HMI and CF can be useful to manage SDR by using a different machine over network.

- **Waveform operations:** These are waveform management operations and they are executed by the operator over HMI and these operations cannot be started before core framework takes start.

We have measured the time elapsed at each SDR life cycle operation. Table 5.13 shows the performance analysis of SCARI-Open core framework.

Table 5.13. Core Framework Life Cycle Timings

Component	Time (sec)
Start CF	1.6
Start HMI	2
Stop HMI	0.5
Stop CF	4.9

Table 5.14 shows the performance analysis of our waveform life cycle operations on SCARI-Open core framework.

Table 5.14. Waveform Life Cycle Timings

Component	Time (sec)
Install WF	3.5
Create WF	12.5
Start WF	0.1
Stop WF	0.1
Release WF	10
Uninstall WF	6.5

Creating the waveform includes parsing the XML files and deploying, configuring and connecting the waveform components. Therefore it takes the most time when compared to other operations. On the other hand, starting and stopping the waveform takes very negligible time because it only requires to call *start* or *stop* method on each component over CORBA.

In the next scenario, we have measured the running performance of our waveform. Implemented waveform has four modes of operation:

- **VoiceTx:** Transmit voice.
- **VoiceRx:** Receive voice.
- **DataTx:** Transmit text or binary data file.
- **DataRx:** Receive text or binary data file.

Figure 5.12 shows the deployment of waveform components over core framework devices and the flow of data between devices and waveform components at each mode of operation.

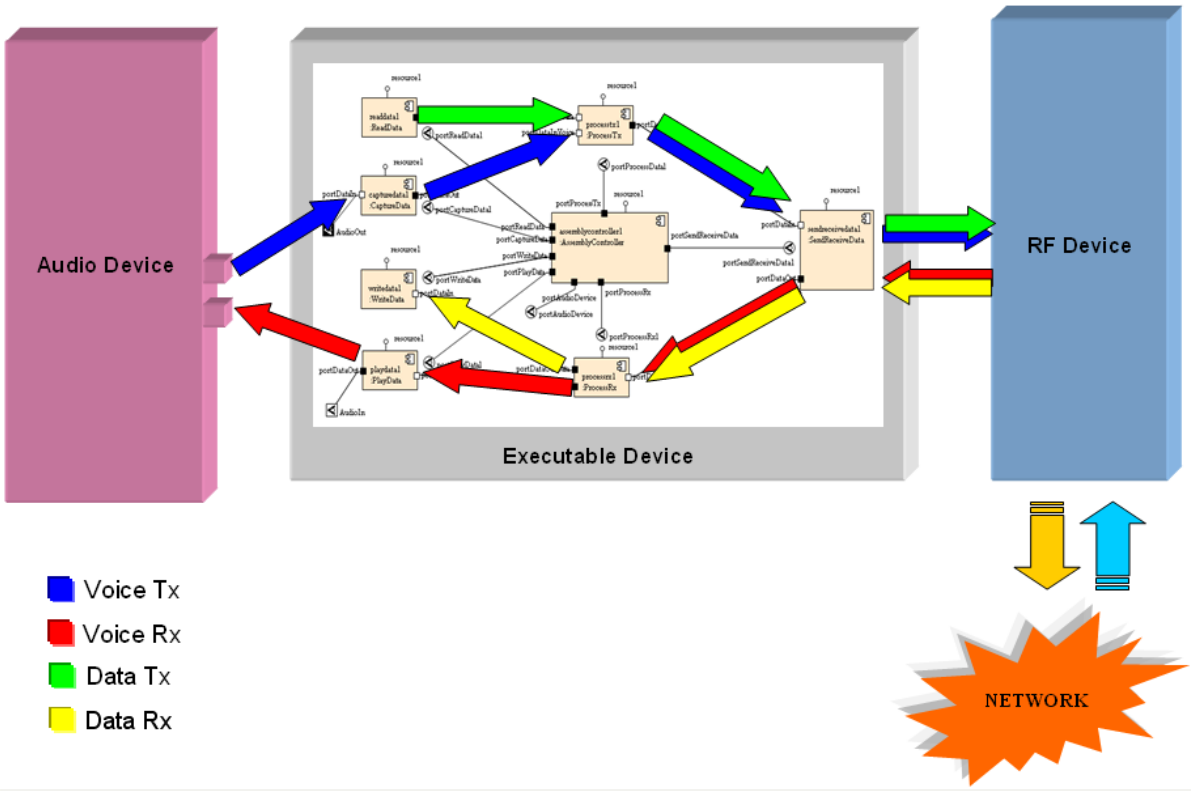

Figure 5.12. Waveform Deployment

In this scenario, we have measured the total time of DataTx and DataRx operations with and without RFDevice. We have tried to see the effect of core framework and middleware overhead for different file sizes. We have also compared results with the time that is required to copy and paste the same files.

Table 5.15 shows the results of our experiments in this scenario.

Table 5.15. Waveform Performance Analysis

File Size (kB)	Time 1 (sec)	Time 2 (sec)	Time 3 (sec)
10	0.4	0.3	0.02
100	3.5	3.2	0.05
1000	32	30	0.1
2000	63	60	0.2
5000	159	155	0.4
10000	322	318	0.9
20000	640	635	1.7
50000	1590	1580	3.4

In this table we compare 3 cases:

- **Time 1:** Data packets travel through ReadData, ProcessTx, SendReceiveData, RFDevice, SendReceiveData, ProcessRx and WriteData.
- **Time 2:** Data packets travel through ReadData, ProcessTx, SendReceiveData, ProcessRx and WriteData. Only difference with case 1 is that packets do not pass over RFDevice, but instead they loop between waveform components.
- **Time 3:** This is the time to copy and paste the same file.

It can be noticed that, the copy and paste method is significantly faster then other two cases, because processing and sending packets between components over CORBA takes too much time. The time required for other two cases does not differ too much, which means removing RFDevice does not decrease total time more than a few seconds even for the largest file. We also observe a linear ratio between file size and the elapsed time which means that overhead of middleware and core framework is constant for each unit of file.

5.2.3. Difficulties Encountered:

We have encountered the following difficulties during development and testing stages:

- Development requires multidisciplinary knowledge including both computer science (C++, Java, CORBA, XML, operating systems, drivers and so on) and communication engineering (Waveform design, SDR hardware, and CR concepts).
- Development takes too much time because there is no debugging facility while developing and testing the waveform. It is because of several reasons:
  - A waveform application consists of several executable files communicating over network.
  - It runs on top of a core framework layer which is also a set of executables possibly in a different programming language as in our case.
  - Waveform implementation may include components running on heterogeneous processing units such as FPGA and DSP.
  - Components of the waveform may run on top of different operating systems.
- Since it is a new subject, the number of existing reference implementations is very small and they have very poor documentation, in addition most of them are highly expensive COTS products.

5.2.4. Evaluation of Using Framework:

Using a framework has the following advantages and disadvantages:

- Programming becomes easier at upper layers in the layered architecture, however flexibility of the programs decreases because of reduced level of APIs. Conversely, programming is more difficult at bottom layers, however the developer is much more free to access and to control the overall system. For example, capturing audio from sound card at driver level is much more difficult compared to capturing audio from core framework, however changing audio device properties is easier in driver level.

- Framework structure, dramatically increases portability by standardizing the interfaces between software layers.
- Performance of the overall system decreases because of increasing level of abstraction.
- Development time of waveforms decreases in layered architectures, because the layers at the bottom provide standard and tested services to upper layers.
- Using framework helps to manage devices and services from a single unit which is useful to manage the whole system.
- Porting legacy codes to the framework may require extra work.

5.2.5. Evaluation of the Metrics:

In this section, we evaluate software engineering metrics of CRs by considering the lessons learned from our waveform implementation.

5.2.5.1. Reconfigurability:. CRs aim automated adaptation to its environment. It requires the radio to be able to change its working parameters at run time. In order to achieve CR concepts, reconfigurability is inevitable and it can only be achieved by developing technologies behind SDRs. The followings can be listed as our evaluations about reconfigurability issues:

- Component based programming increases reconfigurability. This is a result of the fact that it is easier to distribute configuration responsibilities.
- Reconfigurability can be investigated in two ways:
  - **Static reconfiguration:** In this type of reconfiguration, parameters of the executables are saved to configuration files and they are loaded at startup. Modifying configuration files, reconfigures the application, however it requires rebooting. As an example, XML configuration files of our waveform provide static reconfiguring capability.
  - **Dynamic reconfiguration:** Dynamic reconfigurability means to be able to achieve configuration while running. It can be achieved by self reconfigu-

ration of each component or setting configuration parameters over CORBA interfaces. In our waveform implementation, we have 3 types of dynamic reconfiguration property: DestinationID, QoS and ApplicationType. They can be reconfigured at run time by the user interface over network.

  - Reconfiguring GPP based components is easier when compared to DSP and FPGA based processing units. This is a result of the fact that GPP based programs are more flexible and reconfiguring devices such as FPGA requires reboot of the chip and it is not that easy, however there are some newly developing techniques to prevent rebooting.

- Fully automatic reconfiguration of the radio is still far away, because currently reconfiguration mostly requires user intervention. Developing cognitive algorithms is vital to achieve it.

5.2.5.2. Portability:. Portability is an important metric which means running a single implementation on any platform with no modification or with the minimum modification. Portability is a key challenge to reduce development time and cost for SDRs, because otherwise the radio software has to be re-implemented according to any version of the radio. In addition, CRs may download necessary software to be able to adapt it self to the communication standards of the environment. However downloading necessary software does not guarantee that it will successfully run on the target radio if the software is not implemented as portable.

By considering current level of hardware and software technologies and current standardization activities it is very difficult to conclude that true portability is achieved for SDR applications. On the other hand, it does not mean that no thing has been done.

Portability metric has to be evaluated in three categories:

- **Soft components:** Software components running on GPP.
  - Portability is mostly achieved for these components.
  - CORBA provides location transparency.
  - Standardization activities such as SCA provide common interfaces which increases portability.
  - ORB dependent macros or operating system specific codes are the main obstacles among portability.
- **Firm components:** Programmable devices that can load software or firmware such as DSP and FPGA.
  - There are several ORBs for firm components but they are currently at infant ages.
  - Standardization is very poor for these components.
  - There is no simple plug and play capability for these components.
  - They can not directly talk to soft components.
- **Hard components:** Fixed function devices such as hard ASIC.
  - There is no portability for these components. a
  - Using software proxy which abstracts hard components may increase portability.

5.2.5.3. Reusability:. Reusability is also another metric to measure the level of maturity of a software architecture. We have learned the following lessons from reusability studies:

- Dividing an application into smaller units is the key methodology behind reusability.
- Determining the level of reusability is important in order to prevent performance bottlenecks. Because more reusable subunits means more communication overhead among them.
- Analyzing the application requirements and grouping the common tasks are very

crucial to determine the components that has to be reusable.

- Reusability is strictly relevant with portability, because if it is not possible to port the software to other platforms then it is not possible to reuse them.
- Reusability is also dependent on the level of standardization. Determining standard interfaces and protocols between components are crucial to achieve reusability.
- External configuration files make the application more reusable by only modifying them according to the new platforms.
- For GPP based components, achieving reusability is much more easier compared to DSP and FPGA based components.

In our implementation, we have designed our waveform components to maximize reusability. We have defined CaptureData, PlayData, ReadData and WriteData components which can also be reused in future waveforms to provide similar functionality. Also SendReceiveData, ProcessTx and ProcessRx components can be reused after modifying the code to reflect new conditions. On the other hand, it is difficult to reuse AssemblyController component since it is the most waveform specific component. Therefore, it has to be re-implemented in each waveform implementation.

5.2.5.4. Other Metrics:. We can evaluate other metrics briefly as follows:

- **Interoperability:** Interoperability means running different technologies together. In our case, our waveform is implemented with C++ and ACETAO middleware and it is running on a core framework which is implemented on Java programming language and using Java's ORB. This can be an example of interoperability.
- **Scalability:** Scalability means running at different complexity levels. Component based programming and using middleware technologies are the factors to support scalability. As far as each component is implemented as a black box and it talks to other components over well defined interfaces, it becomes easy to insert or remove components to extend or narrow the size of an application.
- **Upgradability:** Upgradability is also similar to scalability. Unit based modeling

has the advantage to make modification on subunits to upgrade them.

- **Realizability:** Standardization of SDRs should define models that are possible to implement. Current SDR standards do not have enough reference implementation to prove their realizability.
- **Affordability:** Current standards are mostly based on open standards such as CORBA, XML and POSIX. So the cost of implementations is reduced.

6. CONCLUSIONS

Considering the current trend of increasing bandwidth usage, we can conclude that effective utilization of the finite radio spectrum becomes a key challenge. CR is build on top of the SDR architecture and promises to solve inefficiencies in spectrum utilization techniques. CR monitors spectrum opportunities and utilizes the bands of the primary users when they do not communicate. This capability requires the CR to make many handoffs during its operation. However, handoff is an expensive operation. It is achieved by suspending the ongoing communication, searching and selecting a new channel, and reconfiguring CR to switch that channel in the shortest time. Therefore, in ideal CR the number of handoffs should be minimized.

In this thesis, we address two main problems. The first one is the channel selection for handoff optimization. We propose two channel selection algorithms and evaluate them for different scenarios. We compare our algorithms with random channel selection and future based optimum channel selection which is the theorical limit. The second problem is the software design issues to cover the needs of SDRs by considering the requirements of cognitive networks. We provide the implementation of a waveform application on an SCA based SDR architecture for evaluating these problems and realizing our algorithms.

Proposed channel selection algorithms consider user connection and spectrum utilization histories to select the channel for making handoff. Collecting the user and spectrum information accurately is critical for the success of our algorithms. Assuming these information is gathered perfectly, our algorithms perform results converging to optimum channel selection in most of the scenarios.

REFERENCES

1. Akyildiz, I. F. and W.-Y. Lee, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey", *Computer Networks*, Vol. 50, No. 13, pp. 2127–2159, 2006.

2. Mchenry, M., "Spectrum white space measurements", *New America Foundation Broadband Forum*, 2003.

3. Cabric, D., S. M. Mishra, and et al., "Implementation issues in spectrum sensing for cognitive radios", *38th Asilomar Conference on Signals, Systems and Computers*, 2004.

4. Dillinger, Madani, and Alonistioti, *Software defined radio : architectures, systems, and functions*, Wiley, 2003.

5. III, J. M., *Software Radio Architecture*, Wiley-Interscience, 2000.

6. Fette, B., *Cognitive Radio Technology*, Elsevier Science and Technology Books, 2006.

7. III, J. M., *Cognitive radio: an integrated agent architecture for software defined radio*, Ph.D. thesis, KTH Royal Institute of Technology, 2000.

8. Haykin, S., "Cognitive radio: brain-empowered wireless communications", *IEEE Journal on Selected Areas in Communications 23*, pp. 201–220, 2008.

9. Rappaport, S., "The multiple-call hand-off problem in high-capacity cellular communications systems", *IEEE Transactions on Vehicular Technology*, p. 546557, 1991.

10. Rappaport, S., "Blocking, hand-off and traffic performance for cellular communication systems with mixed platforms", *Proceedings of the IEEE*, p. 389401, 1993.

11. *SDRF Cognitive Definitions, http://www.sdrforum.org/pages/, Retrieved on October 10, 2008*, 2009.

12. Buddhikot, M. M. and K. Ryan, "Spectrum management in coordinated dynamic spectrum access based cellular networks", *IEEE DySPAN 2005*, p. 299307, 2005.

13. *Shared Spectrum, http://www.sharedspectrum.com*, 2009.

14. Everitt, D. and D. Mansfield, "Performance analysis of cellular mobile communication systems with dynamic channel assignment", *IEEE Journal on Selected Areas in Communications*, p. 11721180, 1989.

15. Grandblaise, D., D. Bourse, K. Moessner, and P. Leaves, "Dynamic spectrum allocation (DSA) and reconfigurability", *Software-Defined Radio (SDR) Forum*, 2002.

16. Yang, L., L. Cao, and H. Zheng, "Proactive Channel Access in Dynamic Spectrum Networks", *Elsevier Physical Communication*, Vol. 1, pp. 103–111, 2008.

17. Cordeiro, C., K. Challapali, D. Birru, and S. S. N, "IEEE 802.22: the first worldwide wireless standard based on cognitive radios", *IEEE DySpan 2005*, pp. 328–337, 2005.

18. Jing, X. and D. Raychaudhuri, "Spectrum co-existence of IEEE 802.11b and 802.16a networks using CSCC etiquette protocol", *IEEE DySPAN 2005*, pp. 243–250, 2005.

19. Ma, L., X. Han, and C. Shen, "Dynamic open spectrum sharing MAC protocol for wireless ad hoc network", *IEEE DySPAN 2005*, pp. 203–213, 2005.

20. Marias, G., "Spectrum scheduling and brokering based on QoS demands of competing WISPs", *IEEE DySPAN 2005*, pp. 684–687, 2005.

21. Nie, N. and C. Comaniciu, "Adaptive channel allocation spectrum etiquette for

cognitive radio networks", *IEEE DySPAN 2005*, pp. 269–278, 2005.

22. Sankaranarayanan, S., P. Papadimitratos, A. Mishra, and S. Hershey, "A bandwidth sharing approach to improve licensed spectrum utilization", *IEEE DySPAN 2005*, pp. 279–288, 2005.

23. Horne, W., "Adaptive spectrum access: using the full spectrum space", *Telecommunications Policy Research Conference (TPRC)*, 2003.

24. Vuran, M. and I. Akyildiz, "AMAC: adaptive medium access control for next generation wireless terminals", *IEEE/ACM Transactions on Networking*, 2005.

25. Cao, L. and H. Zheng, "Distributed spectrum allocation via local bargaining", *IEEE Sensor and Ad Hoc Communications and Networks (SECON) 2005*, pp. 475–486, 2005.

26. Huang, J., R. Berry, and M. Honig, "Spectrum sharing with distributed interference compensation", *IEEE DySPAN 2005*, pp. 88–93, 2005.

27. Menon, R., R. Buehrer, and J. Reed, "Outage probability based comparison of underlay and overlay spectrum sharing techniques", *IEEE DySPAN 2005*, pp. 101–109, 2005.

28. Peng, C., H. Zheng, and B. Zhao, "Utilization and fairness in spectrum assignment for opportunistic spectrum access", *ACM Mobile Networks and Applications (MONET)*, 2006.

29. Zheng, H. and L. Cao, "Device-centric spectrum management", *IEEE DySPAN 2005*, pp. 56–65, 2005.

30. Brik, V., E. Rozner, S. Banarjee, and P. Bahl, "DSAP: a protocol for coordinated spectrum access", *IEEE DySPAN 2005*, 2005.

31. Raman, C., R. Yates, and N. Mandayam, "Scheduling variable rate links via a

spectrum server", *IEEE DySPAN 2005*, pp. 110–118, 2005.

32. Zekavat, S. and X. Li, "User-central wireless system: ultimate dynamic channel allocation", *IEEE DySPAN 2005*, pp. 82–87, 2005.

33. Zhao, J., H. Zheng, and G.-H. Yang, "Distributed coordination in dynamic spectrum allocation networks", *IEEE DySPAN 2005*, pp. 259–268, 2005.

34. Zhao, Q., L. Tong, and A. Swami, "Decentralized cognitive MAC for dynamic spectrum access", *IEEE DySPAN 2005*, pp. 224–232, 2005.

35. Steenstrup, M., "Opportunistic use of radio-frequency spectrum: a network perspective", *DySPAN 2005*, pp. 638–641, 2005.

36. Liu, M., Z. Li, X. Guo, and E. Dutkiewicz, "Performance Analysis and Optimization of Handoff Algorithms in Heterogeneous Wireless Networks", *IEEE Transactions on Mobile Computing*, Vol. 7, 2008.

37. Taha, Hassanein, and Mouftah, "Vertical Handoffs as a Radio Resource Management Tool", *Elseviers Computer Communications*, Vol. 31, pp. 950–961, 2008.

38. Taha, Hassanein, and Mouftah, "Exploiting Vertical Handoffs in Next Generation Radio Resource Management", *IEEE International Conference on Communications*, Vol. 5, pp. 2083–2088, 2006.

39. Chakravorty, R., P. Vidales, K. Subramanian, I. Pratt, and J. Crowcroft, "Performance Issues with Vertical Handovers Experiences from GPRS Cellular and WLAN Hot-spots Integration", *2nd IEEE Annual Conference on Pervasive Computing and Communication*, pp. 155–164, 2004.

40. Bernaschi, M., F. Cacace, A. Pescape, and S. Za, "Analysis and Experimentation over Heterogeneous Wireless Networks", *1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pp. 182–191, 2005.

41. Pahlavan, K., P. Krishnamurthy, and A. Hatami, "Handoff in Hybrid Mobile Data Networks", *IEEE Personal Comm.*, Vol. 7, pp. 34–47, 2000.

42. Mishra, S., A. Sahai, and R. Brodersen, "Cooperative sensing among cognitive radios", *IEEE International Conference on Communications*, 2006.

43. Sahai, A., N. Hoven, and R. Tandra, "Some fundamental limits on cognitive radio", *In Forty-second Allerton Conference on Communication, Control and Computing*, 2004.

44. Sahai, A., R. Tandra, and N. Hoven, "Opportunistic spectrum use for sensor networks: the need for local cooperation", *Information Processing in Sensor Networks*, 2006.

45. Cabric, D., S. M. Mishra, and R. W. Brodersen, "Implementation issues in spectrum sensing for cognitive radios", *Asilomar conference on signals, systems and computers*, 2004.

46. Challapali, K., S. Mangold, and Z. Zhong, "Spectrum Agile Radio: Detecting Spectrum Opportunities", *International Symposium on Advanced Radio Technologies (ISART)*, 2004.

47. Ganesan, G. and Y. G. Li, "Cooperative spectrum sensing in cognitive radio networks", *IEEE DySPAN 2005*, 2005.

48. Ghasemi, A. and E. S. Sousa, "Collaborative spectrum sensing for opportunistic access in fading environments", *IEEE DySPAN 2005*, 2005.

49. Kim, H. and K. G. Shin, "Adaptive MAC-layer Sensing of Spectrum Availability in Cognitive Radio Networks", Technical report, University of Michigan, 2009.

50. Mishra, S., A. Sahai, and R. Brodersen, "Cooperative sensing among cognitive radios", *IEEE ICC*, 2006.

51. Zhang, N. and J. Holtzman, "Analysis of Handoff Algorithms Using Both Absolute and Relative Measurements", *IEEE Trans. Vehicular Technology*, Vol. 45, pp. 174–179, 1996.

52. Marichamy, P., S. Chakrabarti, and S. Maskara, "Performance Evaluation of Handoff Detection Schemes", *IEEE Region 10 Conf. Convergent Technologies for the Asia-Pacific (TENCON 03)*, Vol. 2, pp. 643–646, 2003.

53. Lee, C., L. Chen, M. Chen, and Y. Sun, "A Framework of Handoffs in Wireless Overlay Networks Based on Mobile IPv6", *IEEE J. Selected Areas in Comm.*, Vol. 23, pp. 2118–2128, 2005.

54. Mohanty, S. and I. Akyildiz, "A Cross-Layer (Layer 2 + 3) Handoff Management Protocol for Next-Generation Wireless Systems", *IEEE Trans. Mobile Computing*, Vol. 5, pp. 1347–1360, 2006.

55. McNair, J. and F. Zhu, "Vertical Handoffs in Fourth-Generation Multinetwork Environments", *IEEE Wireless Comm.*, Vol. 11, pp. 8–15, 2004.

56. Fang, Z. and J. McNair, "Optimizations for Vertical Handoff Decision Algorithms", *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC 04)*, Vol. 2, pp. 867–872, 2004.

57. Chang, B. J., S. Y. Lin, and Y. H. Liang, "Minimizing Roaming Overheads for Vertical Handoff in Heterogeneous Wireless Mobile Networks", *Proc. Intl Wireless Comm. and Mobile Computing Conf. (IWCMC 06)*, pp. 957–962, 2006.

58. Acharya, P. K., S. Singh, and H. Zheng, "Reliable open spectrum communications through proactive spectrum access", *TAPAS*, 2006.

59. Shallit, J., *A Very Brief History of Computer Science*, University of Waterloo, 2006.

60. Karatsuba, A. and Y. Ofman, "Multiplication of Many-Digital Numbers by Au-

tomatic Computers", *USSR Academy of Sciences*, 1962.

61. *SoftwareArchitectures.com, Intro to Software Quality Attributes*, 2006.

62. System, J. T. R., "Software communications architecture specification Final Version 2.2.2", Technical report, Space and Naval Warfare System Center, 2006.

63. *JTRS, http://enterprise.spawar.navy.mil/body.cfm?type=c&category=27&subcat=60*, 2009.

64. *SDR Forum, http://www.sdrforum.org/pages/aboutTheForum/aboutTheForum.asp*, 2008.

65. *OMG, http://www.omg.org*, 2009.

66. Group, O. M., "The Common Object Request Broker: Architecture and Specification", Technical report, Object Management Group, 1995.

67. Bicer, S. M., *A Software Communications Architecture Compliant Software Defined Radio Implementation*, Master's thesis, Northeastern University, 2002.

68. "MDA Guide Version 1.0.1", Technical report, OMG, 2003.

69. *Object Management Group (OMG), Software Radio DSIG, swradio.omg.org*, 2002.

70. "UML Specification version 1.1", Technical report, OMG, 1997.

71. *ISO/IEC 15287-2-2000 IEEE Std 1003.13-1998*.

72. Stallman, R., "POSIX 1003.1 FAQ Version 1.12", Technical report, IEEE, 2006.

73. *Open Base Station Architecture Initiative BTS System Reference Document Version 2.0*, 2008.

74. *CPRI, http://www.cpri.info*, 2009.

75. *SCARI 2.2, http://www.crc.gc.ca/en/html/crc/home/research/satcom/rars/sdr/products/scari_open/scari_open*, 2009.

76. *SCARI-Open, https://www.crc.ca/en/html/crc/home/research/satcom/rars/sdr/products/scari_open/scari2_downloads*, 2009.

77. *OSSIE, http://www.mprg.org/research/ossie/download_page.html*, 2009.

78. *GNU Radio, http://gnuradio.org*, 2009.

79. Halperin, D., T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel, "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses", *IEEE Symposium Security and Privacy*, pp. 129–142, 2008.

80. *USRP, http://www.ettus.com/downloads/usrp_v4.pdf*, 2009.

81. *HPSDR, http://hpsdr.org*, 2009.

82. *FlexRadio and PowerSDR web site, http://www.flex-radio.com*, 2009.

83. *SDR Forum Members, http://www.sdrforum.org/pages/currentMembers/ currentMembers.asp*, 2009.

84. Polson, J., "Cognitive Radio Applications in Software Defined Radio", *SDR Forum Technical Conference*, 2004.

85. *SDRForum Cognitive Radio Work Group, http://www.sdrforum.org*, 2009.

86. Cabric, D., S. Mishra, and R. Brodersen, "Implementation issues in spectrum sensing for cognitive radios", *38th Asilomar Conference on Signals, Systems and Computers*, p. 772776, 2004.

87. Jondral, F., "Software-defined radio-basic and evolution to cognitive radio", *EURASIP Journal on Wireless Communication and Networking*, 2005.

88. Briasco, M., A. F. Cattoni, G. Oliveri, M. Raffetto, and C. S. Regazzoni, "Sensorial Antennas for Radio-Features Extraction in Vehicular Cognitive Applications", *SDR Forum Technical Conference*, 2006.

89. Zumbul, A., G. Bozkurt, and T. Tugcu, "Software Defined Radio Architecture for Cognitive Radio", *IEEE SIU 2008*, 2008.

90. Gamma, E., R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

91. Fowler, M., *Writing Software Patterns*, Addison-Wesley, 2006.

92. Fowler, M., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002.

93. Freeman, E., E. Freeman, K. Sierra, and B. Bates, *Head First Design Patterns*, O'Reilly Media, 2004.

94. *Go For Experts Web Site, http://www.go4expert.com/forums/printthread.php?t=5127*, 2009.

95. *Sacha Krakowiak "What's middleware?". ObjectWeb.org.*, 2005.

96. Middleware, S. O., "J. Hurwitz", *DMBS 11.1*, 1998.

97. *Mercury Computer Systems CORBA training presentation for TUBITAK*, 2005.

98. *IEEE Std. 1042-1987 IEEE Guide to Software Configuration Management*, 2009.

99. Bray, Tim, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fourth Edition) - Origin and Goals", Tech-

nical report, World Wide Web Consortium, 2006.

100. *W3C, http://www.w3.org*, 2009.

101. Isler, D., B. Yilmaz, A. Zumbul, and T. Tugcu, "An Entire Architecture for Cognitive Radio Networks", *IEEE SIU 2008*, 2008.

102. Zumbul, A., H. Ozer, and T. Tugcu, "Software Communications Architecture Compliant FM Waveform Implementation", *IEEE SIU 2008*, 2008.

103. Crnkovic, I., S. Larsson, and M. Chaudron, "Component-based Development Process and Component Lifecycle", *Journal of Computing and Information Technology 13 (4)*, pp. 321–327, 2005.

104. *Zeligsoft Component Enabler, http://zeligsoft.com/tools/zeligsoft-ce*, 2009.

105. Zumbul, A. and T. Tugcu, "Applying Design Patterns to SCA Implementations", *SDR Forum Technical Conference*, 2008.