A General Purpose Processor Component Based Software Radio Engine

ABSTRACT

Software radio promises to reduce the dependence on dedicated wireless hardware by using software-defined digital techniques rather than once off dedicated hardware designs. Highly reconfigurable software radio nodes will have a large role to play in mobile communication systems of the future and the possibilities for exploitation of their capabilities are immense. This paper describes a generic software radio engine (SWE) that uses XML to define wireless schemes and platform capabilities and that runs on a general purpose processor platform. The resulting software radio system has a very high level of reconfigurability and can adapt to changing conditions with ease. The complete system also readily facilitates the design of new software radio components and systems.

1 INTRODUCTION

Third Generation (3G) mobile communication systems are set to become widely available over the next number of years. However, there is already much debate about the future viability and success of 3G. 3G is based on traditional ideas of network infrastructure and ownership and on the centralist model of a public carrier network and licensed spectrum. These ideas are already being challenged and it is becoming clearer that the telecommunications industry may have to face radical change with the imminent loss of guaranteed income from regulated infrastructure.

When one looks beyond 3G and focuses on mobile communication systems of the future, two conflicting views emerge. There are those that believe despite the difficulties that 3G already faces, future wireless systems should evolve naturally from 2nd and 3rd generation systems. And there are those that believe future generation mobile communications systems are an opportunity for a fresh start without taking on the history of 2nd and 3rd generation technologies and licensing models.

We are taking the second approach and over the past number of years we have developed expertise around core concepts that we believe will form major constituent parts of any future communication systems. Our view of the network of the future is of a network with an IP core, which supports many different wireless access technologies, with excellent mobility support, that is free from unnecessary operator linkage and that supports end-to-end security. This infrastructure has at its centre such technologies as ad-hoc networks, multiparty-micropayment schemes, peer-to-peer systems and self-organising distributed group dynamics [1].

In terms of wireless communication, a node operating in a 4th generation network must therefore be capable of adapting to the characteristics of the underlying network. A 4th Generation node must be able to reconfigure itself and change to adapt to different wireless technologies, to deal with both infrastructure and ad-hoc systems, to compensate for signal conditions, to adapt to application demands, to adapt to differing regulatory requirements in different locations, to change based on security demands, to reconfigure itself to provide new services, to update to new standards etc. This type of reconfigurability is needed at all levels from the application layer to the physical layer. We believe that software radio is an enabling technology in providing the flexibility and adaptability needed to deal with the demands of mobile networks of the future and that it has a significant role to play in the design of 4th Generation wireless nodes.

Software radio has become a reality due to advances in semiconductor and microprocessor technology. It is
now possible to develop re-programmable devices that can perform multiple functions that previously required separate pieces of dedicated hardware. This area of work is referred to as software defined radio (SDR). SDR systems are typically implemented using dedicated DSP and A/D chipsets.

The work described in this paper looks beyond SDR and at general purpose processors (GPPs) and operating systems. While recognising that DSPs offer good performance, low-cost and power efficiency, there are advantages to working with GPPs. These advantages are well documented [2][3], but it is specifically the high and complex level of reconfigurability that is possible and the ease at which designs can be created, explored and tested that makes the general purpose processor an ideal platform for this work.

The general purpose processor platform allows the design of software radios that are highly reconfigurable, adaptable and flexible. The design process itself is facilitated with ease as software radio components can be implemented in familiar languages such as C and C++ and designers can use the familiar approaches of object-oriented programming and debugging to develop real-time software radio systems.

This paper presents work carried out in designing a software radio system that can meet the requirements of a 4th generation node, that runs on general purpose processors and that is highly reconfigurable and adaptable. Section 2 details the software radio engine and XML loader. Section 3 gives some examples of software radios produced using the system. Section 4 contains an overview of how the software radio fits in the bigger picture of 4th Generation Systems and section 5 concludes.

2 THE SOFTWARE RADIO ENGINE

As stated above the key to software radio in the future is ultimate reconfigurability of the node to adapt to whatever the demand. Reconfigurability both at initialization and runtime is of interest in terms of exploiting the possible uses and roles of software radio in networks of the future. It is also a key concern to be able to create, explore and test designs and ideas with ease. The software radio engine (SWE) outlined here meets these requirements. It has at its core the ability to abstract software radio implementations from underlying hardware capabilities and has been designed to allow for maximum reconfigurability. It comprises of three modules as shown in Figure 1, a repository of signal processing components, an XML loader and a software radio runtime environment.

![Software Radio Engine](image)

Figure 1. Conceptual Overview of the Software Radio Engine

2.1 Repository of signal processing components

The repository of signal processing functions is a collection of components that can be used to create a software radio. The software radio engine allows the creation of different software radios by connecting together different signal processing functions to form the specific wireless scheme of interest.

All signal processing functions are implemented as components that conform to a specific interface defined by the SWE. The use of reflection tags permits the querying of the component at runtime to get information about that component (e.g. data types handled, dynamic properties) so that once a component is developed it is easy to use.

A code pre-processor can be used to generate the components and accompanying documentation describing the functionality of the component.

Each component (e.g. channel extractor, FIR filter) exposes a set of properties that can be used to configure the component. There are two types of properties, normal and dynamic. Dynamic properties can be changed at runtime. For example the cutoff frequency of a filter if set to dynamic could be changed while using the software radio configuration of interest.

Within the component structure, encapsulation can be used to create new components from groups of existing ones. Additional components can also be added to the repository dynamically at runtime thus offering an even great degree of flexibility.

2.2 XML interface

The XML interface is responsible for all interaction with the software radio interface. It performs two main functions.
Firstly the XML interface provides a mechanism for users to describe different wireless schemes using XML. An XML document containing a list of components from the repository of the signal processing components is used to describe the software radio scheme of choice. Changing the document will create a different software radio implementation. The XML document is used as an input to the software radio engine to create the chosen software radio implementation. The XML interface is responsible for priming and connecting all signal processing components according to the structure defined in the XML document. XML documents can be written when needed or a selection can be stored on the node (or at other locations).

Secondly, the XML interface produces XML describing the processing capabilities and components of a particular software radio engine. This facilitates a type of service discovery whereby the capabilities of a SWE can be determined and evaluated. As different SWEs will have different hardware, processing capabilities and sets of components, different SWEs will be capable of running a specific wireless scheme (i.e. offering a particular service). This information allows users of the SWE to determine what services an SWE has on offer. It is of course possible also that ‘missing’ components, needed for a specific scheme, could be retrieved from other locations on the network.

### 3.3 Software Radio Runtime Environment

The software radio runtime environment is responsible for the execution and control of a particular radio configuration (as defined in the XML document). It controls the flow of data and signals through the software radio and as such provides services to all signal processing components. It can use hardware specific knowledge to optimize the performance of a SWE.

For example, signal processing components are implemented by categorizing their function into either CPU bound or I/O bound types. CPU bound components contain CPU intensive algorithms that usually operate on blocks of fixed sized data, e.g. an FFT or FIR. I/O bound components usually process variable sizes of data that may be in some way time dependant, thus requiring multiple threads and blocking for I/O operations, e.g. data acquisition or audio playback. By grouping components in this manner the runtime is able to optimize the use of operating system resources such as threads and events. This produces an overall performance boost in code execution and reduces system latency.

The software runtime environment also facilitates the verification of the particular software radio implementation chosen and provides error information if the configuration requested is invalid.

### 3.4 Implementation

The software radio engine presented is implemented on Windows 2000 in C++. To date, a total of 26 signal processing components have been implemented both for the development of wireless schemes and the analysis of data in the system. The hardware used for the system consists of a high speed PCI analogue to digital converter, custom IF amplifier and RF frontend. An IF of 10.7MHz is sampled using band-pass sampling at 4MHz, transferred to the PC and the remainder of receiver functions are performed in software.

### 3. EXAMPLES

To illustrate how the software radio engine works two examples are explained and discussed.

#### 3.1 FM Receiver

Consider a simple FM receiver whose block diagram is shown in Figure 2.

![Block Diagram of Radio Scheme](image)

Figure 2: Block Diagram of Radio Scheme

Figure 3 shows the FM receiver in terms of its radio components. In order for this software radio to be created the repository of signal processing functions must contain the components needed. Figure 3 consists of six main components, a data acquisition component that controls the flow of data from the hardware frontend to the PC, a channel extractor component that extracts the band of interest from the incoming wideband signal, an FM demodulator, a low pass filter, a downsampler and a component to handle the audio output. The channel extractor illustrates the feature of encapsulation whereby a new component is made from existing components. Any components that are created must follow the guidelines for standard interfaces.
The software radio is created and run by describing the scheme shown in Figure 3 in an XML document. Figure 4 shows an outline example of such a document.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<softwareradio>
  <structure>
    <rfcomponent type="datacapture">
      <parameters>
        <samplinglerate>4000000</samplinglerate>
        <outputblocksize>524288</outputblocksize>
        <channel>0</channel>
        <frequency>104.4</frequency>
        TRUNCATED...
      </parameters>
    </rfcomponent>
    <rfcomponent type="channelextractor">
      <parameters>
        TRUNCATED...
      </parameters>
    </rfcomponent>
    <rfcomponent type="fmdemodulator">
      <parameters>
        TRUNCATED...
      </parameters>
    </rfcomponent>
    <rfcomponent type="lowpassfirfilter">
      <parameters>
        TRUNCATED...
      </parameters>
    </rfcomponent>
    <rfcomponent type="downsampler">
      <parameters>
        TRUNCATED...
      </parameters>
    </rfcomponent>
    <rfcomponent type="audiooutput">
      <parameters>
        TRUNCATED...
      </parameters>
    </rfcomponent>
  </structure>
</softwareradio>
```

Figure 4. XML Document describing a particular Software Radio Scheme

As pointed out in section 3, parameters of the components can be labelled as dynamic and altered during runtime. For example the centre frequency of the FM demodulator can be altered if set as dynamic.

A number of components have also been written to help with the analysis and evaluation of the various software radio schemes. One such component is a spectrum analyser component and can be inserted after any component. For example if inserted before and after FM demodulation it is possible to see the corresponding signals before and after demodulation. This is simply achieved by including this component in the XML document in the appropriate position. The output of this component before and after the demodulator is shown in Figures 5 and 6.
3.2 Automatic Modulation Detection Component

It is also possible to implement more complicated schemes. A significant body of work has been carried out in the area of automatic modulation detection [4]. For example in a packet based network it is possible to alter the modulation scheme used on a per packet basis (to deal with worsening signal conditions for example) using a software radio. This is the type of scheme that would not be possible to easily implement in hardware only and is useful for illustrating the advantages of software radio. An automatic modulation detection system is needed to demodulate the received signal to avoid the wasteful use of extra headers on the packets to indicate modulation type.

The repository of components contains automatic modulation detection components that allow an incoming signal to be analysed and fed to the appropriate demodulation component. The automatic modulation detection scheme is based on a modified moment-based modulation recognition algorithm, designed for robust real-time software radio operation using a GPP. This technique incorporates signal plane partitioning, counting of received signal points within cells and pattern recognition approaches. The result is an improved technique with a minimal processing time overhead that still allows the software radio to maintain its real-time objectives [4]. Figure 7 shows a schematic an automatic demodulation component. The software radio engine allows for parallel components to exist in any software radio scheme and therefore can handle this type of demand. In Figure 7 the output of the modulation detection component is fed to the appropriate demodulation scheme.

The testbed consists of hardware (wireless nodes) and software (a modular communication stack). The system facilitates the creation of a wireless network on handheld WinCE or laptop devices using a variety of radio frontends. The core of the system consists of a ‘generic layer’ interface that allows the dynamic assembly of a network communication stack consisting of the relevant hardware and software elements. The Win32 operating environment has a native implementation of threads and this allows the implicit multitasking inherent in logical layered architectures to be naturally represented in code. The highly modular layered framework allows entire building-blocks to be assembled to suit the network characteristics.

The available hardware frontends are 802.11, a UHF propriety radio and IrDA. Depending on what wireless system is in use, each node of the network is configured with the appropriate layers, including the appropriate physical layer to communicate with the hardware frontend. The system itself can also adapt to changing networks and as such is reconfigurable. Figure 8 shows a node of such a network. It is part of an ad hoc network [6] and is running over a wireless LAN 802.11 system. The ad hoc routing is facilitated by an ad hoc routing layer. In this particular case the ad hoc routing algorithm of choice can be changed if needed. In Figure 8 the system can reconfigure itself to use the Dynamic Source Routing (DSR) protocol, the Ad hoc on Demand Distance Vector (AODV) protocol or the Zone Routing Protocol (ZRP).

4 RECONFIGURABILITY AT LARGE

One of the purposes of creating such a flexible and adaptable wireless node at the expense of other features such as power consumption is to fully explore the possibilities for flexible, adaptable mobile networks of the future. To experiment on a network level with 4G mobile systems a wireless networking testbed was designed [5].

![Diagram](image)

Figure 7 Automatic Demodulation Component

![Diagram](image)

Figure 8 Communication Stack for Testbed

The software radio engine described here can be used to replace the physical layer of this wireless network platform with a software radio layer. The stack interfaces with the software radio layer as can be seen in Figure 9.
This system facilitates the investigation and analysis of very flexible nodes. Reconfigurability of these nodes is possible on a macro and micro level.

5 SUMMARY

This paper focuses on a software radio engine that abstracts the implementation of software radio wireless schemes from underlying hardware. This is achieved by using XML to define the structure and operating parameters of signal processing components. The platform is a general purpose processor platform and the resulting software radio engine is extremely reconfigurable and flexible. The software radio also easily fits in to a highly modular wireless network testbed.

REFERENCES


