

Strategic Support Systems for Crisis Management: A Literature Review

Elandaloussi Sidahmed, Zaraté Pascale

Abstract

This paper presents a literature review of Strategic Decision Support Systems (SDSS) used in various fields such as transport, trade, logistics, medicine and education. The main objective of these systems is to provide information to decision makers to mitigate various influences. The chosen case study of this monograph is COVID-19 crisis management, which is an example that has an impact on sectors such as health, education, economy, the environment, and others. It aims to identify critical dependencies and how to develop efficient solutions. In particular, this monograph explores the problem of support for the strategic planning decision making during COVID-19 crisis management.

Introduction

Software-defined radio (SDR) is an important technology that underlies many modern wireless communications systems for both telephony and data communications. With the availability of cheap high-speed computing platforms, inexpensive electronics, and good software development platforms, we believe that SDR is one of the most effective ways to meet the increasing demand for low-cost, flexible data and voice communication systems.

One of the great things about SDR for undergraduate education is that the technology is popular among hobbyists (particularly amateur radio operators) and academics (many of whom are hams). Thus, a variety of SDR systems can be constructed with relatively low cost and from publically available (i.e., non-proprietary) hardware *and* software components. Examples include the GNU radio[1]and FlexRadio's line of SDRs[2].

SDR is a flexible platform that can be easily modified to suit a wide variety of applications, ranging from changing waveforms, to operational frequency bands, and to specialized user interfaces for particular services. Ideally, only software needs to be changed to make these modifications; there is no need to change the hardware. In addition, since SDR systems (to varying degrees) place most of the radio processing in software, an SDR system can be ported to various computing platforms in a straightforward way. For example, at Grove City College (GCC) we are working on an SDR that will work across various Microsoft platforms: Windows, Windows Mobile, and (we hope) Xbox and Zune devices.

From the software side, an SDR is a self-contained, embedded software system with real-time deadlines and hardware interfaces. The hardware interface for our systems is minimally an RF down-converter and A/D for receiving, and a D/A and an up-converter for transmitting. While the hardware circuitry is well defined, it can be challenging to build because radio operating frequencies are in the radio frequency (RF) spectrum. For example, our applications are in the 0.1 MHz – 50 MHz range, specially, the amateur and commercial radio bands with both voice and data communications. Real-time deadlines occur because the RF signal must be processed as it is received, meaning that streaming buffers need to be consumed and filled without any dropouts, pops, and other distortions.

At GCC, we have an on-going undergraduate research project in SDR. Recently, the students in that project developed a general coverage SDR receiver for decoding commercial AM and FM[3]. That project is being extended to a new operating system and new hardware.

Building on our undergraduate research experience, like that of other schools[4], we launched a senior project in SDR. This project is focused on developing a Digital Radio Mondiale (DRM) receiver[5]. The CS team works in cooperation with an EE senior project team, where the CS team is developing the software and the EE

team is developing a frontend.

SDR Architecture

Figure 1 shows the basic SDR components.

The SDR requires a hardware frontend that contains a “down-converter,” which converts the RF signals at the received frequency into two parts: the I signal (in-phase) and Q (quadrature) signal, which is 90 degrees out of phase (relative to I). To perform down-conversion, we use a Tayloe detector[6]. The detector is a simple, inexpensive circuit that does a complete quadrature downconversion. The I and Q signals feed directly into the soundcard of the PC, where they are converted from analog to digital signals using the soundcard’s A/D converter.

Once converted by the soundcard, I and Q signals are demodulated. This process consists of the following basic steps for receiving[7-10].

1. Time-domain shift: while I and Q are in the time domain, their (center) frequencies are shifted to baseband.
2. FFT: to demodulate these signals, they must be converted to the frequency domain using a Fast Fourier Transform (FFT).

3. Filtering: the signals are low-pass filtered to extract the needed frequency components and to remove redundant information.
4. Shift: the FFT bins are shifted to prevent attenuation at the ends of the signal.
5. Filtering: the signals are high-pass and low-pass filtered so that only the frequencies carrying the information we want are in the FFT bins.
6. Shift: the FFT is shifted back to baseband. This shift is necessary to prevent distortion around the beginning and end of the spectrum.
7. IFFT: an inverse Fast Fourier Transform (IFFT) is performed on the data to get it back to time-domain I and Q components.
8. Demodulation: various methods are used to demodulate the signals. For example, an AM signal is demodulated simply by taking $\sqrt{I^2 + Q^2}$

Given the sampling rate of the soundcard and the types of waveforms we are converting, the software must process each digital sample within about 250 mS.

Pedagogy

The CS department has a two-semester senior project course track. The first semester, a one-credit course is given that emphasizes design (particularly user interface analysis and testability), project planning, specification, and the ethical implications of the students’ software systems. The second semester, a two-credit course that emphasizes implementation, project management, testing, and usability assessment is

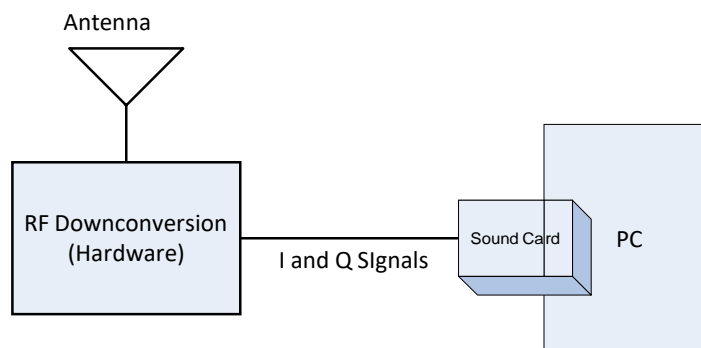


Figure 1: The components of a simple SDR receiver.

given. These courses are required of all CS students. The EE department has a similar senior-project course track, but their courses have different emphases, particularly in testing and implementation.

We give the ABET outcomes for the two CS senior project course and comment on SDR's application to those outcomes. These things are found in Table 1. The CS and EE faculty believe it is important for students to gain experience in multi-disciplinary projects. SDR provides a perfect project around which to organize such multi-disciplinary CS and EE teams for the following reasons:

- The software and hardware components are easily divisible.
- The interfaces between the components are well-known and are (fairly) straightforward; yet allow some interesting design choices.
- Since there are existing hardware and software platforms available (see Summary for more information), the CS students can proceed with their development and testing while the EE team designs and fabricates their hardware. The EE students can test their hardware using existing SDR software to ensure it works before final testing with the CS students' software. This "division of concerns" makes the projects more management for both teams. More importantly, the ultimate success of each group is not necessarily tied to the other group. This helps to prevent internecine conflicts among the students (and faculty!).
- The SDR applications are very broad, from amateur radio to public service (e.g., police, fire, and other emergency services) to specialized applications (third-world communication). For students with a strong service inclination, SDR is a natural fit.

CS Students

Our senior CS students have taken at least four programming courses, in addition to algorithms, data structures, and software engineering. They are experienced with advanced object-oriented design

and large-scale team-based programming. In addition, most CS students gain experience with the Windows programming model, .Net, XNA, DirectX and COM. In our game-programming courses, they gain experience with real-time programming, particularly meeting hard deadlines and the tradeoffs between on-line and off-line processing. SDR gives students the opportunity to use the programming and algorithms knowledge they acquired during their years of study.

Furthermore, CS students get experience with low-level implementation details, something that is often missing in typical CS undergraduate education. They need to pack and unpack floating point numbers represented in a variety of bit formats (e.g., byte-by-byte interleaved), as well as interface directly to Windows's audio streaming buffers.

EE Students

By their senior year, EE students have a good background in signal processing with additional courses in communications and control theory. At this point, they also have completed courses in analog and digital circuit design, as well as microprocessor interfacing. With this background, they are able to design the radio frontend.

Project Organization

The EE and CS teams met occasionally over the two terms as needed (mostly, as prodded by the faculty). In these meetings, we discuss both progress and various technical problems and how to solve them. Mostly, this involves ensuring the interfaces are correct and coordinating project management items. By and large, the teams work independently.

The complexity of the projects, both hardware and software, takes all the efforts of each group. Because of this, the work stays divided by discipline. It is not practical for CS majors to learn enough about circuit design and fabrication to actively participate in the EE's tasks, and the same holds for EE students participating in software design, implementation and testing.

Table 1: ABET Course Outcomes.

Outcome 1: How to design and evaluate, from multiple perspectives, a software system.	Since SDRs are used by non-computing professionals, an easy-to-use interface is essential. SDR systems must be evaluated from a usability perspective. Moreover, algorithm performance and hardware performance (e.g., energy use) must be considered. Finally, these systems must conform to FCC regulations, which expose students to regulatory compliance. (We do not do this compliance analysis, as we use FCC-approved transmitters).
Outcome 2: How to present and defend ideas in front of a group.	Students must present their design and rationale, as well as system specifications and final product demonstration and accompanying documents to technical and non-technical faculty at end of each semester. In addition, frequent demos and technical walkthroughs give students opportunity to present and defend their systems before classmates. Since most CS students in the project courses do not know SDR concepts, the “SDR students” learn how to explain these things to their classmates.
Outcome 3: How to use software-engineering methods for implementing and testing a software system	SDR systems are complex and can only be built by a group. Multidisciplinary teams force each team to interact with those outside their expertise. The testing experience is richer than software-only systems (ditto for hardware-only systems), as they test against hardware that is evolving with their software.
Outcome 4: How a software team is organized and operates	The SDR experience enriches the usual project management components of senior design projects, as the software team must also accommodate a hardware team that runs in a different way than they do. For example, hardware fabrication takes weeks and is (typically) one-shot, this is very different from the nearly instant code-build-refactor methods used by software teams.
Outcome 5: How to search and read articles in the computing literature	The SDR project forces students to read a variety of literature outside what they normally read. CS students must read about hardware systems, digital signal processing, and radio engineering.

Thus, the SDR project is a true multidisciplinary one, where the students must learn to work with each to get the complete system built.

Results

The CS and EE students each finished their respective projects. The EEs designed the circuitry and created a multilayer circuit board design. The board was fabricated by an external manufacturer. The board was then populated by the students and a staff technician. Unfortunately, a mistake in the circuit board design prevented the front end from operating properly, although, portions of the board did operate as designed. There was no time for the team to get another board fabricated.

The CS students were able to get a functioning AM and FM SDR operating in the commercial radio bands. They were not able to get a DRM receiver working, mostly because of the complexity of decoding the DRM signal.

Since the EE’s board did not operate, the two teams were not able to integrate their projects; the CS team used a commercial receiver. While it was disappointing to both teams that they could not complete the entire project, both teams were able to significant components of their own systems working.

We ran similar projects in the past several years, once as an EE-only senior project in SDR and once as a multidisciplinary team. The projects’ results were mixed: both projects were able to get AM and FM decoding working, but not in real time. These systems were able to process recorded I and Q

signals of arbitrary duration. The problems both groups faced were hardware and soundcard interfacing. While the circuits appear to be simple, there are practical issues with relatively high-frequency circuit layout that cause novice designers problems. Interfacing to the soundcard is more complex than it seems at first: unpacking and scaling the output of the card's A/D converters requires considerable skill.

Summary

The SDR project provides undergraduate students the opportunity for multidisciplinary projects. The project technology underpins today's advanced wireless systems, and it reinforces the classroom knowledge gained by both computer science and electrical engineering students.

The SDR project is fun, and it is great practical experience. The project gives students the chance to work with students in another discipline, allows them to apply much of what they learned during their matriculation, and exposes them to an important technology.

We note that CS departments can easily use SDR as a software-only project simply by purchasing the necessary hardware components. A variety of companies produce hardware that will supply I and Q signals (see Flexradio[2], TenTec Radio Mondiale Receiver[11], and the cost-effective, hobby-oriented SoftRock[12]). In addition, EE or CE programs can use SDR as a hardware-only project by using existing open-source software systems, such as the GNU radio project, Flexradio's software, or Dream for Radio Mondiale[13].

References

1. Blossom, E., *Listening to FM radio in software, step by step*. Linux Journal, September 2004, Vol. 125.
2. FlexRadio Systems. [Online] <http://www.flex-radio.com/>.
3. Birmingham, W. and L. Acker, *Softwaredefined radio as an undergraduate project*. Proceedings of ACM SIGCSE, Covington, KY, 2007.
4. Silage, D., *Reintroducing Amateur Radio In ECE Capstone Design Projects*. Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition.
5. Digital Radio Mondiale Consortium. [Online] <http://www.drm.org/>.
6. Tayloe, R., *US Patent #6,230,000*
7. Youngblood, G., A Software-Defined Radio for the Masses, Part 1. *QEX*. 2002, Vol. July/Aug, pp. 1-9.
8. —. A Software-Defined Radio for the Masses, Part 2. *QEX*. 2002, Vol. Sept/Oct.
9. —. A Software-Defined Radio for the Masses, Part 3. *QEX*. 2002, Vol. Nov/Dec.
10. —. A Software-Defined Radio for the Masses, Part 4. *QEX*. 2003, Vol. Mar/Apr.
11. Ten-Tec . [Online] <http://radio.tentec.com/>.
12. SoftRock. [Online] <http://www.softrockradio.org/>.
13. Dream DRM Receiver. [Online] http://apps.sourceforge.net/mediawiki/drm/index.php?title=Main_Page.