Utilizing Open Source Software For Military Applications

Soldier's Personal Situation Display

A Case Study of Rapid-Prototyping using Axiomatic Design and Component Oriented Software Engineering

By
Jeff B. Smith

A MASTER OF ENGINEERING REPORT

Submitted to the College of Engineering at
Texas Tech University in
Partial Fulfillment of
The Requirements for the
Degree of

MASTER OF ENGINEERING

Approved

_____
Dr. Atila Ertas

_____
Dr. T.T. Maxwell

_____
Dr. M.M. Tanik

_____
Dr. Chris Letchford

October 21, 2006

ACKNOWLEDGEMENTS

I would like to take a moment to express my gratitude to Raytheon and all those involved in the Texas Tech Systems Engineering program for providing me the opportunity to further my education. It was a great honor and the effort wasn't wasted. I intend to utilize my new skills by doing everything within my means to help make Raytheon the best defense contractor in the world.

Secondly, I would like to thank my family for their patience and support. Very special thanks go to my wife Laura and my daughters (Kady, Carlee and Michael) for taking up all my slack. Thanks for carrying on without Dad for a year. "I'll make it up to you – I promise ". My sincere hope is that my children now understand and appreciate that you never ever stop learning - even an old guy like me.

Last but not least, thanks to the entire Texas Tech staff for assembling such a phenomenal program. If I were to sit down and design the perfect master's level engineering program for working professionals – I expect I'd arrive at what you have assembled. Thanks don't begin to express my feelings. In any event – Thanks to Professors Ertas, Maxwell, Tate, Tanic, Yee, Dogru, Vaughn, Green, Iyengar and all the others who took valuable time to invest in my education.

Table of Contents

DISCLAIMER

This document was prepared as a requirement for graduation with Masters of Engineering from Texas Tech University. While this document is believed to contain correct information, no warranty expressed or implied, exists regarding the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or claims that its use would not infringe upon privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by myself, Raytheon, or Texas Tech University. The views and opinions contained within are expressly my own and do not represent Raytheon, Texas Tech or the United States Government.

ABSTRACT

As a case study or exercise using Rapid-Prototyping, Axiomatic Design (AD) and Component Oriented Software Engineering (COSE), I propose to design and develop a prototype unit I am calling the Soldier's Personal Situation Display (SPSD). This exercise is in partial fulfillment of the requirements for the award of the Masters of Engineering degree in Systems Engineering from Texas Tech University.

It is also my intent to discuss in detail all aspects of using, modifying and leveraging Open Source software in a Military/Industrial environment. I intend to highlight and discuss what Open Source software is, the pros and cons, benefits, problems and lessons learned while using Open Source software from an engineering standpoint (specifically as applied to Rapid Prototyping using Axiomatic Design and Component Oriented Software Engineering). I will focus on my experiences in designing and building the prototype unit using Axiomatic Design, Component Oriented Software Engineering with open source software and commercial off the shelf hardware.

Table of Figures

List of Tables

Chapter 1

Introduction

As United States combat Soldiers and Marines are deployed more and more frequently to geographically restricted and hostile urban terrain, the importance of each war-fighter being cognant of his or her immediate situation and surroundings becomes apparent. One can use as a textbook example, the events that took place in Mogadishu, Somalia, where U.S. Army Rangers and Special Forces units were severely hampered in their ex-filtration from a mission by a lack of orientation and situational awareness, as well as a lack of intelligence and the location of their opposing forces. On the other hand, in some cases the war-fighter may be overwhelmed with information and have no clear way to quickly process the incoming data. These problems call for some sort of automated assistance for the war-fighter in rapidly processing, analyzing and utilizing digital intelligence.

A similar incident where my proposed hardware (Soldier's Personal Situation Display) unit would have helped avoid if not actually prevented casualties is the case involving the former NFL professional football player turned U.S. Army Ranger – Pat Tillman. As you may recall Pat Tillman was killed in action by so-called "friendly fire" in Afghanistan after his squad was separated from their convoy while traversing a particularly dangerous section of road known to be frequented by hostile forces. Pat Tillman's squad of Rangers was taken under fire mistakenly by members of his own platoon, when they (Tillman's unit) were misidentified as an enemy combatant force. Had members of both U.S. Army units been equipped with my proposed hardware (SPSD), they would have been able to pinpoint the exact geographical location of both their own unit and Tillman's unit to within a few meters via Global Positioning System (GPS) coordinates – thus avoiding mistakenly firing on members of their own forces.

CHAPTER 2

BACKGROUND

Friendly fire (also known as fratricide, non-hostile fire, or *blue-on-blue* engagement) is a term defined as an attack on friendly forces by other friendly forces. It may be either deliberate (incorrectly identifying a target as the enemy) where fire brought to bear on enemy forces accidentally ends up hitting one's own troops; or due to errors of identification where friendly troops are mistakenly attacked in the belief that they are enemy forces (such as the Pat Tillman incident). So-called "friendly fire" incidents such as this were common during the first and second world wars, where troops fought in close proximity to each other and targeting was relatively inaccurate. In general, highly mobile battles and battles involving troops from many nations are more likely to result in this kind of incident. The following statistics involving friendly fire incidents are provided by the Pentagon:

- World War II: 21,000 (16%)
- Vietnam War: 8,000 (14%)
- Gulf War: 35 (23%)
- Invasion of Afghanistan (2002): 4 (13%) [1]

High casualty rates in Iraq and Afghanistan from Improvised Explosive Devices (IEDs) point to soldiers having inadequate situational awareness. Recent research has focused primarily on detection of IEDs, but I intend to approach the problem from a different direction - IED Avoidance. By leveraging the following prior research:

---

[1] Fatalities were Canadian Soldiers, not American. Caused when a US fighter pilot dropped a 500 lb bomb while Canadian soldiers were performing a live fire exercise on April 17, 2002

- **Smart Agent Technology**
    - Peters, Look, Quigly, Shrobe, and Gajos [1]
    - Dolev, Gilbert, Schiller, Shvartsman and Welch [2]
    - Gray, Kotz, Cybenko and Rus, Dartmouth, 1998 [3]
    - Wendleken, McGrath, Blike [4]
    - Brewington, Gray, Moizumi, Kotz, Cybenko, Rud [5]

- **Component Oriented Software Engineering / Axiomatic Design**
    - Dogru and Tanik [6], Dogru [7]
    - Suh and Doh [8]
    - Repenning, Ioannidou, and Payton [9]

- **Open Source Software Integration**
    - Joe Jacob [10]
    - Martin, Cheyer, and Moran [11]
    - Mitre Corp [12]

The above noted areas of research ("Smart Agents", Component Based Engineering, and Open Source Software) all deal in theoretical design or applications. I intend to follow-through to the next logical step and develop a "real" system, leveraging and utilizing preexisting open-source software and technology based on the previous mentioned research. Smart Agent and general AI (Artificial Intelligence) systems exist primarily in the lab or hosted onboard large processors. It is my intent to push the technology down the chain to the actual soldier in the field, automating and connecting what are now autonomous technologies.

Chapter 3

DESCRIPTION

3.0 Description

"If you load a mud foot down with a lot of gadgets that he has to watch, somebody a lot more simply equipped – say with a stone ax – will sneak up and bash his head in while he is trying to read a vernier."

-- Robert Heinlein, Starship Troopers

The *Soldiers Personal Situation Display* (SPSD) will provide known locations of all friendly forces operating in the immediate area via the use of individual Global Positioning Receivers, each transmitting its own unique location to peers within an encrypted, ad-hoc wireless (802.11) network, shared ONLY among the operators/end-users of the SPSD(s). Admittance into the ad-hoc wireless network shall be via password and all transmitted data shall be required to be strongly encrypted via freely available open source software.

The SPSD will also provide the dismounted infantryman an embedded Knowledge Based (KB) System using "Smart Agents" that will assist the soldier in making tactical decisions (avoiding hazardous areas, etc.). Prior research [13] has shown that soldiers currently lack the skills and equipment to quickly and accurately record and report intelligence worthy information. The SPSD will help alleviate this problem. Says one U.S. Army officer, Major Ray: "We had lots of patrols going out, but very few reports coming back and some of their reports were not going into the system." Collin Agee, director, intelligence, surveillance and reconnaissance integration at the U.S. Army G-2 office, notes that, out of 400,000 patrols conducted in the Iraq War, only 6,000 reports reached brigade level. The G-2 (intelligence) assessment was that rear-area soldiers were not trained to do reporting and were not focused

on situational awareness input. Hence there is ample evidence a need for a device like the Soldiers Personal Situation Display exists.

Using both Axiomatic Design and Component Oriented Software Engineering (COSE) techniques, it is my intent to rapid-prototype the SPSD, which will be a portable, handheld digital assistant much like the Personal Digital Assistants (PDAs) many of us currently use today (Blackberry, Palm-Pilot, etc).

Figure 1: Soldier using PLGR GPS [Rockwell Collins]

I also intend to restrict my choices of software to Open Source/Commercial off the Shelf (COTS) components exclusively in order to fully exploit and demonstrate the benefits of open-source software within the COSE model.

The Soldier's Personal Situation Display will provide the war-fighter with a small, portable, handheld, digital graphic display unit with touch-pad input. It will provide a visual, GPS-driven geographic situation display of the soldier's current position, as well as the current position of other friendly units in the immediate, selectable area. It will provide a data logging and recording function allowing the dismounted infantryman to capture and record intelligence data or other information for later play-back (i.e. After Action Reports/AAR). It will also provide a simple point-to-point messaging service

(ala AIM or "Instant Messenger"). In closing it will provide the following:

- ✓ **Small, Cheap and Handheld!**
- ✓ **Basic Navigation Function via Global Positioning System (GPS)**
- ✓ **Provide Location of "Friendly" forces within the wireless network**
- ✓ **Agent-Based [Artificial Intelligence] Soldier's Assistant**
- ✓ **Basic Communication and Proximity Warning Function**
- ✓ **Provide Wireless 802.11 Access**
- ✓ **Basic Data Reporting/Recording /Recalling Function**
- ✓ **Integral Onboard Encryption**
- ✓ **Linux/Open Source based Software**

Table 1: Informal SPSD Functional Requirements

3.1 Rapid Prototyping with Open Source Software

"Rapid-Prototyping" is defined as the engineering process of quickly putting together a working model or prototype in order to test various aspects of a design, illustrate ideas or features, and gather early user feedback. Using rapid prototyping techniques was fundamental to the design process of the Soldiers Personal Situation Display as the core effort of the development of the SPSD was integrating COTS (Commercial-Off-The-Shelf), open-source software packages onto the core processor as quickly as possible, with minimal effort and zero or minimal induced software bugs. Limiting the core software to preexisting open-source software packages and modules limited risk and cost and enabled the leveraging of tested, debugged software.

The integration of the software onto the core CPU was staged iteratively in order to enable testing, verification and performance in multiple cycles. This enabled errors, deficiencies in design or faulty integration practices to be discovered early and quickly corrected. When the prototype SPSD was sufficiently tested and met the functional requirements (robustness, manufacturability and other design goals) the unit was deemed ready for evaluation. The benefits of following a rapid-prototyping

development cycle while developing the SPSD were:

- ✓ **Prototypes are easily modified**
- ✓ **Provides a proof of concept necessary to attract funding**
- ✓ **Early visibility of the prototype gives users an idea of system behavior**
- ✓ **Higher probability of end-user satisfaction**
- ✓ **Encourages participation among users and producer**
- ✓ **Enables a faster delivery of working system**
- ✓ **Cost effective (Development costs reduced)**

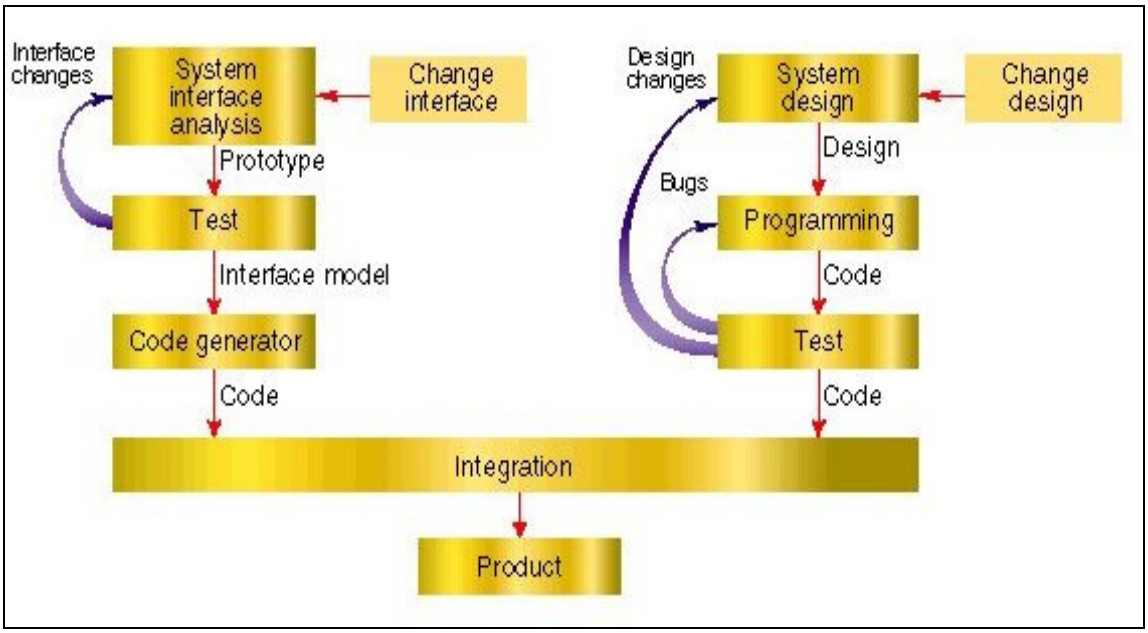

Figure 2: Rapid Prototyping Process [Dr. Dobbs's Journal]

3.2 Commercial off the Shelf Components

A Commercial Off-The-Shelf or (COTS) component is a product that is used with little or no modification, is easily integrated, and designed to operate with existing system(s) essentially as-is. Added benefits of utilizing COTS based software are low cost and high quality. A good example of a COTS

software component would be software bought by the computer user i.e. word processing and office suites such as “OpenOffice” or operating systems (Linux). There are further subtypes of **COTS** components such as **GOTS**, **MOTS**, **NOTS** or **NDI**. The following subtypes of COTS products are defined:

I. A *Government Off-The-Shelf* (GOTS) component is a product that is developed and provided by the government agency for which it was created. It may be either developed by the agency itself or developed by a third party under contract from the agency to develop the component based on specifications provided. When dealing with the government or government agency as a customer, the acquiring agency may prefer to require GOTS components, as the government typically maintains control over all aspects of GOTS components.

II. A *Modifiable Off-The-Shelf* (MOTS) software component is a COTS component whose source code can be altered or modified. Another name for Open Source software might by a MOTS component. The component may be altered or modified by the end-user to fulfill the requirements of the specifications. Different licensing rules may apply to MOTS software (GNU GPL license, Free Software License (FSF), or other).

III. A *Niche Off-The-Shelf* component or (NOTS) product refers to vendor-developed, typically proprietary software that is designed and developed for a specialized, narrow market, as opposed to the broad market for COTS products.

IV. *Non-Developmental Item*. An NDI is any previously developed component used exclusively by the contracting customer, and is typically not commercially available. Furthermore it covers any component that requires minor modifications or modifications of the type customarily available in the commercial marketplace in order to meet the functional requirements of the customer’s specifications.

Carney and Long [2] define a usability modification matrix helpful in visualizing the inherent complexity of (re)using COTS-based software products (Figure 3), where *Source* is defined as the origin of the component, and *Modification* is defined as degree of modification of the component (possible or required).

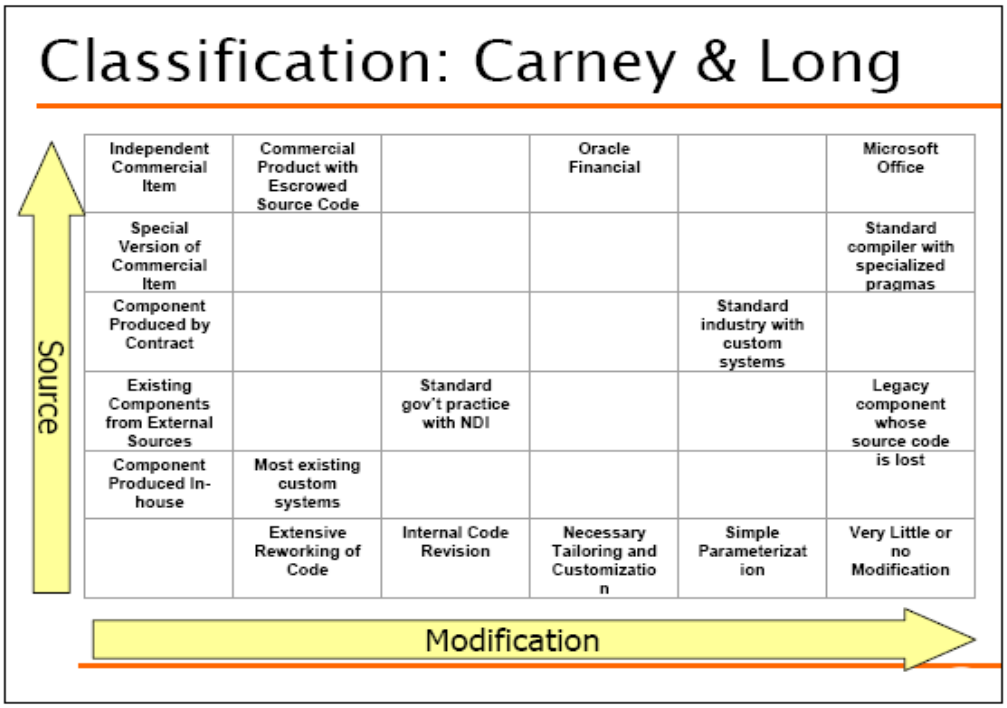

Figure 3: COTS-based Software Complexity Reuse [Carney and Long]

[2] Carney and Long, "So What do You Mean By COTS?", IEEE Software March 2000

3.3 Axiomatic Design

By definition - Axiomatic Design or AD is a systematic, scientific approach to design. It guides designers through the process of first breaking up customer attributes or needs (CAs) into functional requirements (FRs), then further refining these requirements into design parameters (DPs), and finally deriving a process which produces those design parameters with the help of process variables or methods (PVs), thus creating a logical progression through the four domains of the design world: Customer, Functional, Physical, and Process as seen in Figure 4.

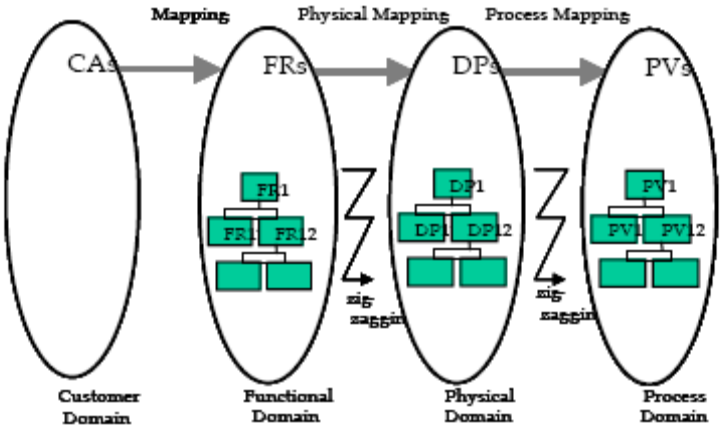

Figure 4: Axiomatic Design Process [3]

3.4 Component Oriented Software Design

Component Oriented Software Engineering (COSE) assumes the availability of preexisting software entities, which may or may not be object-oriented in nature, and are currently available for the designer to leverage and use – much like commercial off the shelf hardware components. Just as an electrical engineer might pick and choose from a collection of capacitors, diodes, op-amps, etc. to build a

circuit card assembly, the software engineer might select a set of components such as a signal-processing software module or a data encryption software module to assemble a functioning software package. See Figure 5.

A common example of a software component might be a browser plug-in that enables the end-user to view an image or video (MPEG), or listen to a particular song or piece of music (MP3, WAV, etc.)

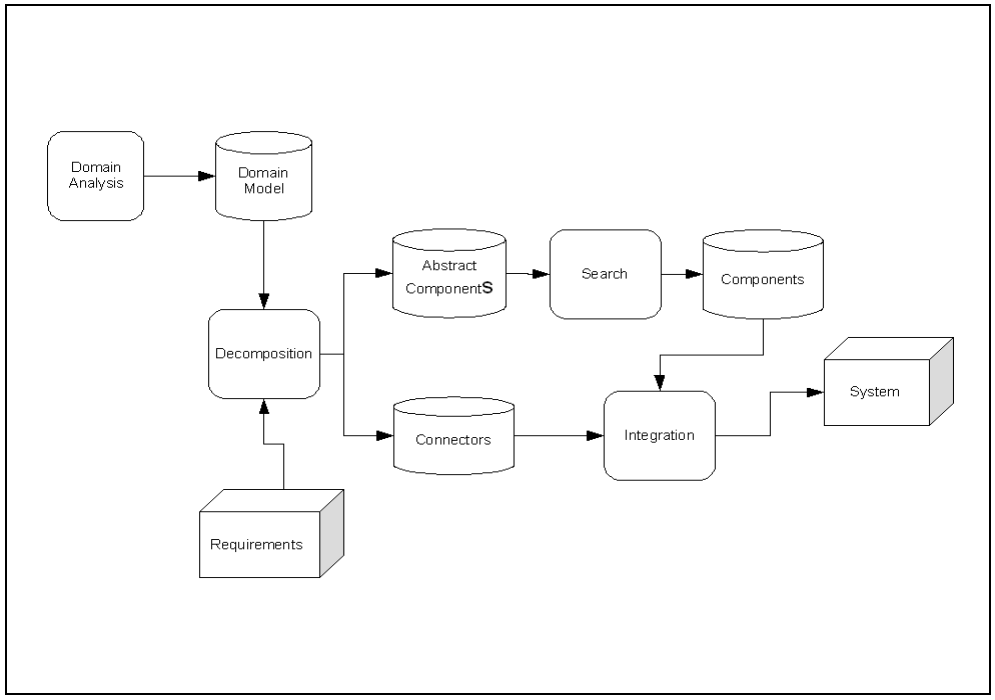

Figure 5: Component Oriented Engineering Process [7]

3.5 Global Positioning System

What is GPS? The Global Positioning System is a space-based constellation of satellites combined with a network of terrestrial ground stations used for monitoring and control, collectively used for radio-navigation. An array of satellites orbits the Earth providing accurate information on position,

---

[3] Basem, "Axiomatic Quality: A Framework for Axiomatic Design and Robust Integration", IDPT 2005

velocity, and time anywhere in the world and under any weather condition.

The GPS satellite system originally developed by the United States Department of Defense is officially named NAVSTAR (Navigation Signal Timing and Ranging) and is managed by the 50th Space Wing at Schriever Air Force Base and is available for use for many applications. GPS has become a widely used global utility, used for navigation on land, sea, and air around the world, as well as an indispensable tool for cartography and surveying. GPS also provides as a service, an extremely precise time reference source which is used in telecommunications and wide branches of scientific research.

In late 2005, the first in a series of enhanced capability GPS satellites was added to the orbiting constellation, offering new capabilities, including a second civilian GPS transponder/signal called L2C for improved accuracy and more robust reliability. In the coming years, additional satellites will extend the coverage of L2C and a third and fourth civilian signal are also scheduled to augment the system, as well as provide advanced military capabilities. The Wide-Area Augmentation System (WAAS), available since August 2000, increases the accuracy of GPS signals to within 6 ft for receivers capable of supporting WAAS. The accuracy of GPS provided navigation can be improved even further, to about 1 cm over short distances using Differential GPS (DGPS), which uses an additional fixed GPS beacon to further reduce error.

The GPS constellation consists of 24 satellites in circular orbits, where the orbits are designed so at least four satellites are always within reception of any GPS receiver from most any place on earth. There are four satellites grouped in each of six orbital planes and each satellite circles the Earth twice each day at an altitude of 11,000 miles. The constellation also provides three spare satellites parked in orbit. Each orbit is inclined 55 degrees from the equatorial plane with a right ascension of sixty degrees.

The flight paths of the satellites are measured by five ground control stations located in Hawaii, Colorado Springs CO, Kwajalein Island, Ascension Island, and Diego Garcia in the Indian Ocean. The master control station in Colorado processes the combined observations and sends updates to the satellites

through the other ground stations. The transmitted updates are used to synchronize the atomic clocks onboard each satellite to within one microsecond and also adjust the ephemeris of the satellites' internal orbital model to match the observations of the satellites from the ground.

Each satellite continuously rebroadcasts the exact time according to its own internal atomic clock along with data that includes the orbital elements of the satellite's precise position, satellite status messages, and a relative fix of the approximate position of every other active GPS satellite. This information allows ground based GPS receivers to use data from the strongest satellite beacon to help locate other satellites.

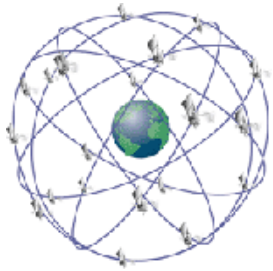

Figure 6: GPS Satellite Constellation [NASA]

3.6 Wireless Networks

Wireless networking is a technology that enables computers to communicate using standard IEEE 802 network protocols, using the RF spectrum as the transmission medium. The most widely used standard is 802.11 produced by the Institute of Electrical and Electronic Engineers (IEEE). This is a standard defining all aspects of Radio Frequency Wireless networking.

There are two kinds of wireless networks. The first, a *peer-to-peer* wireless or *ad-hoc* network which consists of a collection of computers each equipped with a wireless networking interface card or

(NIC). Each computer can communicate directly with all of the other wireless "connected" computers. They may share files this way, but may not be able to access wired Local Area Network (LAN) resources unless one of the computers acts as a "bridge" to the wired LAN using special hardware and-or software.

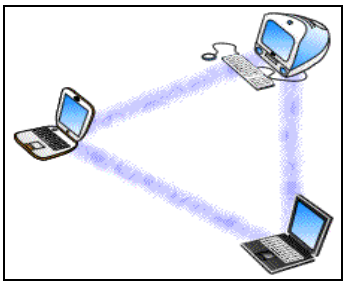

Figure 7: Ad-Hoc or Peer to Peer Network

The second type of wireless network uses an access point, or base station. In this type of network the access point acts like a hub, providing connectivity for the wireless computers. It can connect or "bridge" the wireless LAN to a wired LAN, allowing wireless computer access to LAN resources, such as file servers, remote databases or other remote resources. There are two types of access points:

1. Dedicated Hardware Access Points (**HAP**) – HAP access points provide most of the features of a full featured wired network.

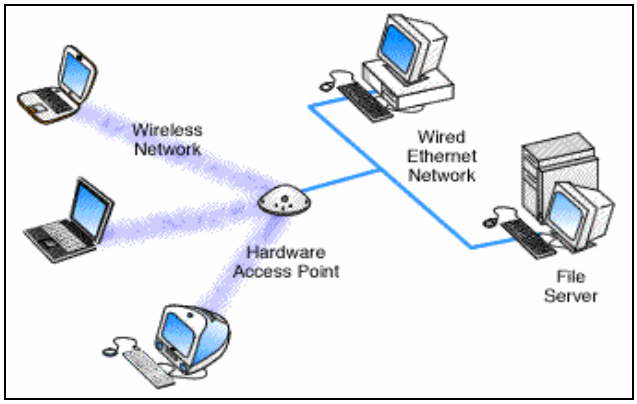

Figure 8: Hardware Access Point

2. Software Access Points (**SAP**s). SAPS are typically computers equipped with a wireless network interface card and used in an ad-hoc or peer-to-peer wireless network. “Software routers” can be used as a basic Software Access Point, and include features not commonly found in hardware , such as direct “Point-to-Point-over Ethernet” or (**PPPoE**), but may not offer the full range of wireless features defined in the IEEE 802.11 standard.

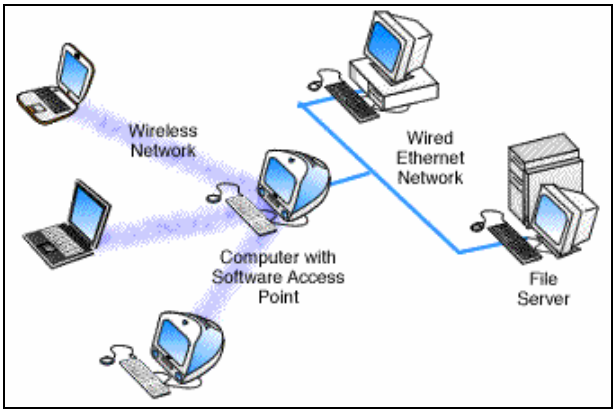

Figure 9*: Network using Software Access Point*

3.7 Agent-Based Technology

A *software agent* is a software component that acts on behalf of another software component or entity and has the authority to decide when and if some predefined action is appropriate. The idea is that agents are not strictly invoked for a task, but trigger themselves to act or perform. *Nwanna* [4] defines the Agent typology illustrated below:

[4] Nwanna, “Software Agents: An Overview”, Knowledge Engineering Review, 1996

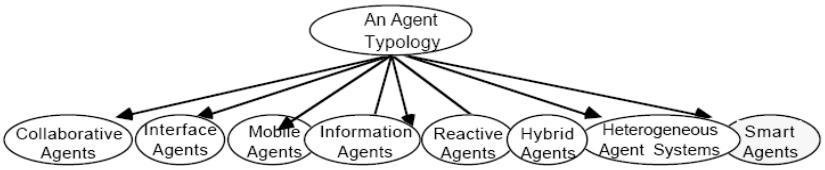

Figure 10: Software Agent Typology

A specialized instance of a software agent is a "*Smart*" agent. Smart agents typically perform tasks requiring learning or reasoning (sometimes defined as "Artificial Intelligence") and assist the user (soldier) acting on their behalf, performing repetitive computer-related tasks. Smart agents typically possess the following characteristics:

- ✓ Ability to adapt in real time
- ✓ Ability to learn and improve through interaction
- ✓ Ability to learn quickly from large amounts of data
- ✓ Ability to accommodate new problem solving rules incrementally
- ✓ Possess memory based storage and retrieval capacities
- ✓ Possess parameters to represent short and long term memory to enable (forgetting)
- ✓ Ability to analyze its own behavior, error and or success.

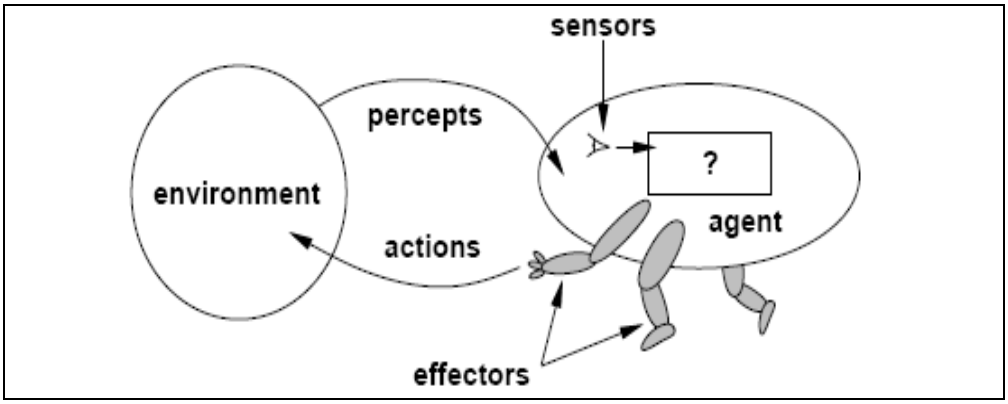

Figure 11: Intelligent Agent [5]

---

[5] Russell, Norvig, "Artificial Intelligence, A Modern Approach", Springer Verlag 1995

A second type of software agent of interest to us is the *Mobile* agent. A mobile agent is a software component that can relocate from processor to processor in a heterogeneous, distributed network. The component decides when and where to migrate. It can put itself to sleep at an arbitrary point, move to another processor and wake itself on the new processor. *Baumann and Hohl* [6] details a mobile agent model where a series of *Service* agents provide an interface for applications to request the help of a mobile agent. (Figure 12)

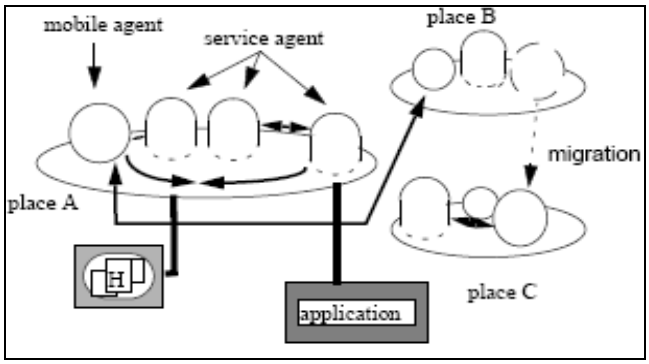

Figure 12: Mobile Agent

The use of intelligent and mobile based agent software technology is fundamental to the implementation of the SPSD. In my design, I propose to use both *intelligent* and *mobile* based agents to assist the soldier, with the agents acting as a "second set of ears, eyes and brains".

3.8 Open Source Software

So what is Open Source? **Open Source software** is any software distributed under a license which allows the end-users to change or share the actual source code itself. Source code is the human readable, pre-compiled text-form of a computer application. By definition, Open Source applies primarily

[6] Baumann, Hohl, "Mole – Concepts of a Mobile Agent System", Univ. Stuttgart, 1997

to source code, although the concept of "openness" has begun to spread to other technologies leading to new genres such as **Open-Hardware** [7] or hardware whose initial specification, usually in a software format, are published and made available to the public for free use, **Open Source Curriculum** defined as instructional resources whose digital source can be freely used, distributed and modified, and even **Open-Cola** - whereas Coke and Pepsi withhold their recipes as closely guarded secrets, volunteers have posted a formula for a similar drink. The taste is said to be comparable to that of the standard beverages.

While there are many different kinds of open source software, all have one universal characteristic: the authors insist that the source code be made available whenever the corresponding application is used, distributed or modified. In this way, open source is almost the opposite of traditional intellectual property paradigms like patents and copyrights, which seek to keep knowledge restricted to the creators or who they choose to sell the knowledge to. In defiance of conventional wisdom and the economics of intellectual property, open-standards have led to the creation of many significant underlying innovations throughout the internet community.

Licensing of open source software is a broad topic. The GNU General Public License or **(GPL**) is the most common and widely used open source software license and was authored by Richard Stallman [8]. According to published accounts, as of April 2004, the GPL accounts for nearly 75% of the over 23,000 free-software projects cataloged on www.freshmeat.net, and about 68% of the projects cataloged on www.sourceforge.net, the two most popular open source projects sites. In general the GPL grants the recipients of a computer source code the following rights:

1) The right to run the open source program for any purpose desired.
2) The right to study the internal mechanisms of the program and how it works and the right to modify it. Obviously access to the source code itself is a precondition for this. Note: This allows open source to be fruitfully mined for component reuse.
3) The right to redistribute your modified copies.

---

[7] www.ax84.com, An Open Source Amplifier

4) The right to improve or enhance the program, and release the enhancements to the public. Access to the source code is a precondition for this.

The current version of the GPL license, version 3, was released in 2006. GPL3 is said to have addressed complaints by businesses and be more "commercial" friendly, although as of this writing the author of the Linux operating system Linus Torvalds has refused to license the Linux kernel under GPL3 and has retained GPL2 [9] as the license for all Linux kernel baselines.

On the other hand**, Closed-Source** or for-profit commercial software applications and packages do not make the source code available; instead they supply and sell the application binaries only (compiled application source code) – of none or little use to anyone wanting to understand how the program works and complicating potential component reuse. The source code of such programs is usually regarded as a proprietary trade secret of the company. Under certain conditions access may be granted to the source code by third parties but typically the authors will require the party to sign a non-disclosure agreement or (NDA).

Critics of open-source argue that its openness is not a virtue but an Achilles' heel. Linux is currently the selected operating system for many future U.S. defense systems, such as Future Combat System (FCS/Army), the Land Warrior, and the Global Information Grid, which is envisioned to connect future military systems into a single heterogeneous network. Critics claim this is cause for serious concern and its security is inadequate for defense use. They claim the systems now under development utilizing open-source components are a national security risk since open-source software is developed under a cooperative effort of globally dispersed engineers (such as Russia and China) who live in countries not necessarily on friendly terms with the United States. They argue that foreign intelligence agencies and terrorists can easily infiltrate the Linux community to contribute subversive software.

---

[8] "GNU Mainfesto" Dr. Dobbs Journal, 1985
[9] www.linux-watch.com

Obviously this is of concern and the well-found arguments should be and are currently being addressed. Examples of this being “Security-Enhanced Linux” developed and controlled by the National Security Agency (NSA) [10]. In closing, it is my opinion that skilled, knowledgeable, highly trained engineers can scour the software and detect any security shortcomings or outright holes.

[10] http://www.nsa.gov/selinux/

CHAPTER 4

System Design and Implementation

4.0 Concept and Problem Statement

Informally, the problem is to develop a lightweight, rugged, handheld device to provide various services and facilities; such as allow military personnel in the field to display and monitor their own geographic position as well as viewing the positions of other personnel also equipped with the mobile display unit (SPSD) (Figure 13), as well as provide a agent-based soldier's assistant providing features such as warning of proximity to dangerous areas, shortest or safest route(s) for infiltration or ex-filtration, shortest route to medical facilities, etc.

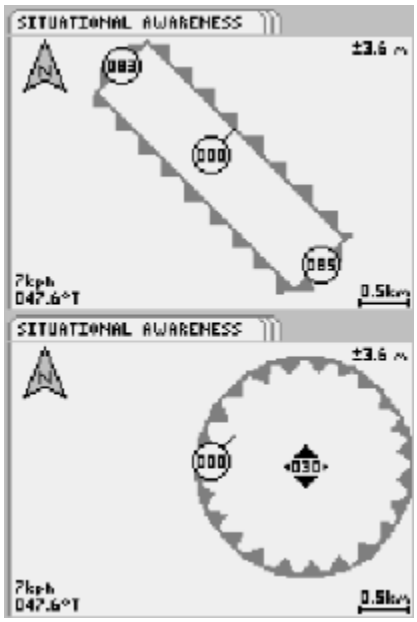

Figure 13: SPSD Example Screens [Rockwell Collins]

Specifically features provided include (primary), the ability to tag via mil-std (FM 101-5-1) [14] symbology, (geographic) areas of interest or potential hazard such as locations of potential IEDs (Improvised Explosive Devices), general hostile areas, closest medical facilities, plot(s) to closest safe havens, etc. Software-based "smart" agents will assist the user in locating or avoiding selected areas of interest. Secondary functions could possibly include language translation, data recording, note taking, image viewing, and soft field manuals FMs in PDF format, (ex first aid, weapons handling, and maintenance or procedures field manuals).

More specifically the problem is three-fold:

1) To develop/prototype/assemble the physical unit itself
2) To devise a method or protocol to exchange GPS position information and other areas of interest with other agents residing on remote SPSD units, as well as displaying shared significant symbology (waypoints, markers, dangerous landmarks or areas),
3) To develop the agent-based software using only open source components (Cougaar[11]), and to port any necessary existing software components to the SPSD unit.

4.1 System Design and Implementation

The high-level SPSD system model consists of four parts:

1) SPSD unit
2) Mobile wireless 802.11 hardware infrastructure
3) GPS module
4) End-user open source software components onboard the SPSD unit.

[11] Open source agent development toolkit, DARPA (www.cougaar.org)

4.2 System Model

4.2.1 Soldiers Personal Situation Display

The actual host CPU for the SPSD was selected and chosen via requirements arrived at by using Axiomatic Design (AD) techniques. One of the major controlling requirements for this project was all hardware and software must be Commercial-Off-The-Shelf and open-source derived. Using AD derived requirements, the processor chosen was a Sharp SL-5500D (Figure 14), as there is a stable open source (GNU) cross compiling (ARM/arm-linux-gcc 2.95) toolset for this processor.

Figure 14: Sharp SL-5500D

CPU -- 206 MHz Intel SA-1110 Strong-Arm system-on-chip processor

Memory – 64MB DRAM and 16MB Flash

Display -- 3.5-inch 240 x 320-dot pixel TFT 65,536 color LCD with touch panel support

Keyboard -- Front lighted QWERTY keyboard with a slide cover

Dual Card Slots -- Compact Flash (CF) type II and Secure Digital (SD)

I/O ports -- IrDA 1.2, Serial and USB

Audio -- Stereo headset jack with audio input (Recording)

Power source -- Lithium Ion battery (950 mAH)

Battery Life -- 10 hrs (backlight off) 1 hr (backlight on)

Size -- 2.9x5.4x.7 in (with keyboard hidden)
Weight -- 6.8 oz

Table 2: Sharp SL-5500D Specifications

4.2.2 Socket Compact Flash 802.11b Wireless Adapter

The SPSD must not only be able to graphically display the user's current geographic position by means of GPS, it also must be able to transmit each individual user's current position to its peers and receive the current positions of its peers via wireless (802.11) and display them on the SPSD/GPS map grid. Using requirements derived from Axiomatic Design, the Socket Communications P500 Compact Flash Wireless adapter card (Figure 15) was chosen to fulfill the wireless communications requirements.

Figure 15: Compact Flash Wi-Fi Card [Socket Technology]

Physical Characteristics:	**Standards Conformance:**
Card Size: 56.1 x 42.8 x 3.3 mm	*Meets 802.1x requirements*
Total Mass: 13.6 g	*Wi-Fi Certified Enterprise Edition*
Inserted portion of card is Type I CF	*WPA2-Enterprise*
Power Consumption (3.3 V Supply):	*CCX v2*
Idle: <20 mA	*IEEE 802.11b and 802.11g*
Transmission: 265 mA (peak)	*Compact Flash Specification 2.0*

Transmit Power:	**Hardware Support:**
CCK: 16.5 dBm typical	*Type I Compact Flash slot*
OFDM: 14.5 dBm typical	*Compact Flash Specification 2.0*

Table 3: Compact Flash Wi-Fi Specifications

4.2.3 Wireless 802.11 Network Infrastructure

One of the most important features if not the single most important feature of the SPSD is providing the capability to both track and view (via GPS) all members/participants of the secure wireless network. The Sharp SL5500D provides the client node interface, while the extended wireless network infrastructure is provided by the Fortress Technology ES520 wireless access bridge. The ES520 was chosen from requirements derived from an Axiomatic Design exercise. *Note: for uses in short, line-of-site distances, the ES520 would be optional as the SPSDs can each operate stand-alone in an ad-hoc network.* The ES520 (Figure 17) provides secure wireless access to the network via 802.11 a/b/g. An "all-in-one" device, the ES520 combines an access point, wireless bridge, switch and secure gateway in a compact, rugged form factor ideal for use in rapid deployment in a tactical network, easily fitted to a HMMWV. See Figure 16.

Figure 16: HMMWV Mobile Communications Platform

Figure 17: ES520 Wireless Bridge [Fortess Systems]

Back-haul Range -- tested up to 32 miles (antenna-dependent)
Access Range -- tested up to 2 miles (antenna-dependent)
Performance -- up to 100 secure clients
Encryption -- AES-128, 192, 256
WPA2 (80211i) software upgradeable
Authentication -- internal or external RADIUS
Management -- secure browser-based GUI, CLI or SNMP
SSID support -- up to 4 SSIDs

Table 4: ES520 Specifications

4.2.4 Open Source Host/Client Software

As discussed in the previous section, the single most important feature of the SPSD is providing the capability to both track and view (via GPS) all members of the secure wireless network. Likewise, the core "essence" of this exercise is proving the feasibility of reusing 100% open-source COTS based software for a rapid-prototyped, tactical military device.

Per the requirements derived initially, the software must provide or support the following: be small and cheap, a basic navigation function via Global Positioning System (GPS), provide location of "friendly" forces within the wireless network, provide basic communication function(s), provide wireless 802.11 access, provide basic data reporting/recording /recalling function, support strong, integral onboard encryption , host an Agent-based "soldier's helper", and most importantly be Linux/Open Source based.

Using Axiomatic Design, the following software "components" were identified for the initial configuration:

Linux Kernel
Qtopia Graphics toolkit/library
OZ -- Open-Zaurus (http://www.openzaurus.org/)
QPE GPS – Open source/Qtopia GPS software (qpeGPS)
QPE MPEG Player – Open source/Qtopia audio record/playback utility
GNU X86/ARM development toolkit (compiler/linker/debugger/GNU 2.95)
Open-Source Wireless Client (QTJim)
Component/Open Source derived Agent-Based Expert System

Table 5: Axiomatic Design derived Software Components

4.2.5 Global Positioning System Module

4.2.5.1 Global Positioning System External Unit

Figure 18: GPS 12 [Garmin International]

Navigation features

Routes: 20 reversible routes with up to 30 points each, plus MOB and Trac-Back modes
Trac-Back: Automatic track log navigation
Map datums: 107 including one User Datum
Coordinates: Lat/Lon, UTM/UPS, plus other grids, including Maidenhead and User Grid

Performance

Receiver: 12 parallel channel receiver continuously tracks, uses up to 12 satellites to compute and update position

Acquisition times: Approx. 15 seconds warm, 45 seconds cold, 5 minutes with Auto-Locate and 45 seconds with EZ-init, easy initialization

Update rate: 1/second, continuous

Accuracy: 15 meters (49 feet) RMS, 1-5 meters (3-15 feet) RMS with GARMIN GBR 21 DGPS receiver (optional) and up to 0.1 knot RMS steady state velocity

Dynamics: 6g's
Interfaces: NMEA 0183 and RTCM 104 DGPS corrections
Antenna: Internal

Table 6: Garmin GPS-12 Specifications

4.2.5.2 GPS Communications Mode

Communications between the external GPS receiver and the SPSD is via RS-232 serial. The Garmin GPS-12 is capable of outputting NMEA [12] 0183 data. The NMEA 0183 Interface Standard defines electrical signal requirements, data transmission protocol and time, and specific sentence formats for a 4800-baud serial data bus. Each bus may have only one talker but many listeners. The Garmin GPS-12 outputs to the SPSD ASCII text NMEA 0183 "sentences" once per second (1 Hz) in the format:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:

GGA Global Positioning System Fix Data

123519 Fix taken at 12:35:19 UTC

4807.038, N Latitude 48 deg 07.038' N

01131.000, E Longitude 11 deg 31.000' E

1 Fix quality: 0 = invalid

1 = GPS fix (SPS)

2 = DGPS fix

3 = PPS fix

4 = Real Time Kinematics

5 = Float RTK

6 = estimated (dead reckoning) (2.3 feature)

7 = Manual input mode

8 = Simulation mode

08 Number of satellites being tracked

0.9 Horizontal dilution of position

545.4, M Altitude, Meters, above mean sea level

46.9, M Height of geoid (mean sea level) above WGS84 ellipsoid

(Empty field) time in seconds since last DGPS update

(Empty field) DGPS station ID number

*47 the checksum data, always begins with *

[12] National Maritime Electronics Association

4.3 Axiomatic Design Detail

Axiomatic Design was developed by Dr. N.P. Suh [15] at MIT, Department of Mechanical Engineering. Dr. Suh was motivated by the lack of scientific design principles and proposed using axioms as the scientific foundation of his design process. Out of the twelve axioms first suggested, Suh concentrated on the following two basic principals for Axiomatic Design:

Axiom 1: The Independence Axiom
Maintain the independence of the functional requirements.

Axiom 2: The Information Axiom
Minimize the information content in a design.

We define the following terms in reference to Figure 4. The set of **Functional Requirements** (FRs) is the minimum set of unique requirements that completely describes the design based on needs of the user/dismounted soldier (also know as **Customer Attributes** or CAs), and the final design product is defined as the derived solution that most elegantly and clearly satisfies those requirements (FRs). This is accomplished through the mapping between the FRs in the functional domain and the **Design Parameters** (DPs) in the physical domain and through the mapping in the process domain between the design parameters DPs and the **Process Variables** (PVs). The final high-level software components required to fulfill the mission of the Soldier's Personal Situation Display (SPSD) will be defined using the techniques suggested by *Suh* for developing software systems [16].

4.3.1 Define Functional Requirements of the Software System

According to Axiomatic Design theory, the first step in designing a software system is determining the **Customer Attributes** (CAs) or requirements the software system must satisfy. Then the Functional Requirements (FRs) and **Constraints** (Cs) of the software are verified that they do indeed

fulfill the customer requirements, where the FRs are defined as the minimum set of independent requirements that the design must satisfy [Suh], and the Constraints (Cs) are defined as the bounds upon the solution(s).

The high-level requirements defined earlier are as follows - small and cheap, provide a basic navigation function via GPS, provide location of friendly forces within a definable geographic area, provide a basic communication and proximity warning function, provide wireless 802.11 access, provide ~ 8 hour battery life, provide basic data reporting/recording /recalling function, support and provide integral onboard robust encryption, host an agent-based system to assist the soldier in decision making, and finally – the software ***must be 100% open source***.

Given the above basic criteria (customer needs), the following Functional Requirements and Design Constraints were synthesized using axiomatic design techniques:

Software must be 100% Open Source based
Unit must be hand held
Unit must be cheap to produce
Unit battery life must be at least 8 hours
Unit must provide 802.11 (Wi-Fi) access
Unit must be rugged
Unit must provide ROBUST encryption facilities!
Unit must have a graphical display
Unit must have touch pad
Bonus if multi-input (touch-pad and keypad)
Unit must be field upgradeable (firmware/software)

Table 7: Design Constraints

.

Handheld Unit to Display and Manage Information
Handheld Unit to Report and Record Data
Handheld Unit to Provide RF Communications
Handheld Unit to Provide Situational Awareness

Table 8: Top Level Functional Requirements

4.3.2 Mapping Domains to Independent Software Functions

This step of Axiomatic Design involves mapping the Function Requirements (FRs) for the SPSD into the actual physical domain (i.e. generating Design Parameters or (DPs)). Axiomatic Design theory refers to this process as "Decomposition by Zigzagging" (see Figure 4.). While Axiomatic Design requires the presence of DPs to mathematically satisfy the coupling/decoupling rules, AD does not itself favor any methods of actually defining the DPs themselves. Other papers and research has been done to address this particular short-coming using such methods as TRIZ, Knowledge Based Engineering, and COSEML in conjunction with AD [17]. *Tanik, Dogru and Grimes* propose using Object Modeling Technique (OMT) and High-Level Architecture (HLA) standards to address the specific case of potential interface incompatibility when using software components with AD. Ultimately, however, the various techniques used for deriving DPs, etc. are beyond the scope of this paper.

The following components were derived to satisfy the functional requirements via "Brainstorming". The initial conceptual Customer Needs (CNs) were driven from my personal experiences in the U.S. Army and current events in the Middle East (Iraq/Afghanistan). Specifically relating to the derived DPs, the Garmin GPS-12 was chosen because I already owned one and had tested it under military conditions, and the unit had a serial interface enabling connection to an external host processor (SPSD).

Garmin GPS-12 Handheld GPS Receiver
Sharp SL-5500D PIM
Wireless client capability(802.11)
802.11 Infrastructure
Open Source (COTS) S/W (Agents to store waypoints (Lat/Lon) of Areas of Interest and software to display AOI's and other hazardous areas[13]***)***

Table 9: Top Level Design Parameters

4.3.3 Define Information Content for Software Systems

According to the Information Axiom of Axiomatic Design [Suh], the design that has the least information content is the best design. Suh defines the information content in terms of the probability of arriving at a design that meets the functional requirements. It can also be calculated in mathematical terms according to the following expression: *Information content (I) = log (1/P)*, where *P* is the probability of successfully satisfying the functional requirements. The *probability of successfully satisfying the (FRs)* is represented as the function of both the design range to satisfy (range designer trying to satisfy), and the capability of the proposed solution known as the system range. *But how is the information content of a design quantified?* Metrics might be measured by number of lines of code, the MIPS rate consumed by the host, or the levels of decomposition and the maximum count of the FRs of the system. Unfortunately, these are not absolute measures of the inherent complexity, but more of a "WAG" in my opinion. Ultimately I made the decision to use very loose tolerances to minimize information content, as this is a Rapid Application Development (RAD) exercise.

4.3.4 Decompose FRs, DPs, PVs, and Software Modules

This step of Axiomatic Design involves breaking down the top-level requirement and design parameters to the lowest level via decomposition where they can be mapped directly to either a software component or a stand-alone piece of COTS hardware that fulfills and meets the requirement (see Figure 19). Before decomposing to a lower level, the DPs must be determined for that level in the physical domain. This iterative process is called zigzagging. It is worth noting that at this phase of design, it is not unusual to reanalyze the original specifications as "over-specifying" requirements may limit design decisions.

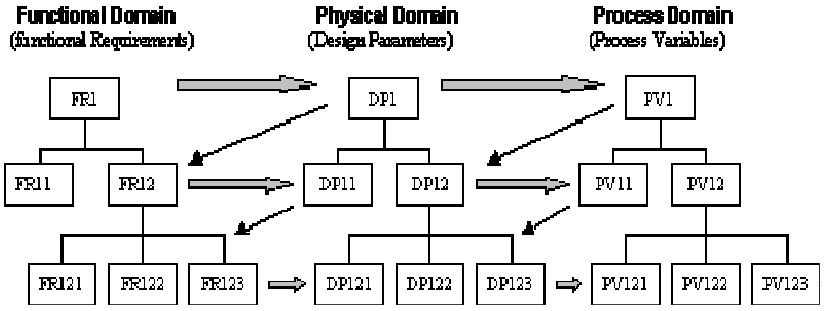

Figure 19: Requirements Decomposition to Leaf Level via Axiomatic Design

As one of the fundamental purposes of this exercise was to discover and experiment with divergent techniques to assist in the fielding of a rapid-prototyped, "quick-turn" produced device, extra attention was paid when decomposing requirements. When rapid-prototyping with the intent of maximizing the use of COTS components, great care must be taken when decomposing high level

---

[13] This software is "To-Be-Developed" open-source derived

requirements. Requirements should be studied closely and reasoning applied as to not decompose a requirement to such a refined (leaf) level, such that a COTS component is unable to reasonably satisfy that requirement. An example might be where a specification existed for a piece of software to provide a service {A}, but a preexisting open source software component was known to exist and provide services {A, B, C}. In some cases it would probably be cheaper and quicker to either modify the existing open source component software or alter your specification as opposed to developing an entire new component. In these cases it is best to obtain the advice of a domain expert when one is available and can assist you. In my case, I made frequent use of a Google-like search engine called **rpmfind** [14] and a centralized web-site containing thousands of open source projects covering different technology domains – **sourceforge** [15]. Using rpmfind I was able to enter the needed term ("wireless"), and all open source packages containing that term were returned (Figure 20). Using the sourceforge site, I was able to quickly locate an open source java based software development package for designing, coding, and debugging agent-based components called **Cougaar** (*Cognitive Agent Architecture* funded by DARPA [16]) . Having access to tools such is rpmfind and sourceforge can be invaluable to the developer when faced with short schedules and quick turn-around times. A domain expert will typically possess the knowledge and whereabouts of tools such as rpmfind, sourceforge and others. Being able to instantly search for and access the needed open source components based on requirements and other criteria proved invaluable.

[14] http://www.rpmfind.net (A search tool for Linux/open source based packages and components)
[15] http://www.sourceforge.net, Central site for all open source projects
[16] Defense Advanced Research Projects Agency

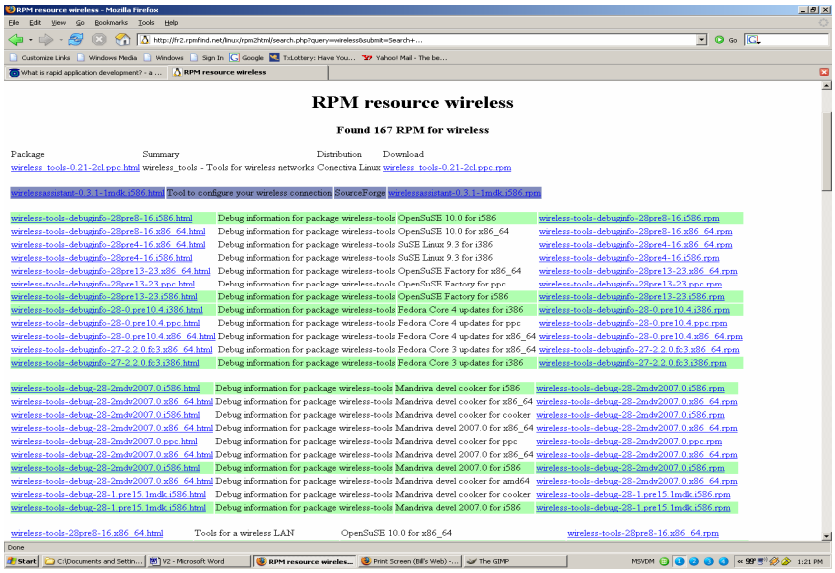

Figure 20: RPMFind Web based search engine

4.3.4.1 Functional Requirement Decomposition

At this point, the Top-Level Requirements are decomposed to the lowest level at which they can be satisfied or *as-close-as-possibly* be satisfied by preexisting open source components or COTS hardware. RPMFind was used predominantly in the location of software component packages. The initial top level design matrices containing refined functional requirements and design parameters are shown below.

Functional Requirements	Design Parameters			
	PDA	***Software***	***GPS***	***Radio***
System to Display/Manage Data	X			
Provide Expert System	X	X		
System to Provide Geographical Orientation	X		X	
System to Provide Basic RF Communications	X			X

Table 10: Top Level Design Matrix

4.3.4.1.1 Data Management Subsystem

The Data Management subsystem onboard the SPSD should provide at least the basic leaf-level functionality of what most other palm-top operating systems. I chose to use the "**Open-Zaurus**"[17] palmtop environment as it meets the basic requirements of:

1) Component software is open source
2) Component software comes packaged with "Make" files for the host target - (Sharp SL-5500D/ARM processor) and therefore requires **no** porting!
3) Component provides all the functionality of the native Sharp supplied OS ROM and more – audio playback and record, note-taker, generic light weight database, web-browser (Opera), image viewer.

Functional Requirements	Design Parameters			
	PDA	***Recording Software***	***Database Software***	***OS/Adapter Software***
System to Categorize/Store Data	X	X		
System to Process Data	X	X	X	
System to Run Applications	X	X	X	X

Table 11: Secondary Data Management Design Matrix

[17] [Open-Zaurus] http://openzaurus.sourceforge.net/

4.3.4.1.2 Agent Based Expert System

The "Expert" subsystem (one of two primary subsystems) onboard the SPSD should provide the soldier with an Agent/Knowledge-Based (AKB) system allowing him or her to quickly access pertinent data or intelligence via an encrypted data path. The AKB system should interface with an internal persistent database and allow the storage and referencing of locations via military grid coordinate (UTM) of hazardous areas, locations of closest medical facilities, safe extraction routes. The onboard AKB should also provide access to field manuals such as (military procedures, repair, first-aid, or weapons handling).

The AKB system could also incorporate a programmable rudimentary language translation facility [18] such as those described by [Losh]. There are currently class room oriented computer-based language trainers [18] but I propose integrating this feature into the SPSD. Current computer/lab based language trainers offer elaborate scenarios involving cultural and nationalistic scenarios that are clearly beyond the capability of a hand-help unit. From my experience, in most cases a simple dictionary translation facility is more than adequate for the dismounted soldier on patrol. Open Source components such as "Babel-Chat" exist that perform rudimentary translations, but were not evaluated. Per Axiomatic Design techniques the basic requirements of the SPSD AKB system are listed in Table 12.

Functional Requirements	**Design Parameters**				
	PDA	***Recording Software***	***Database Software***	***Agent Software***	***Translation Software***
System to Record Logs or Notes	X	X			
Provide a Database function	X		X		
Provide ability to warn soldier of Geographic located hazards	X	X	X	X	
Provide Rudimentary Language Translation	X		X		X

Table 12: Agent-Based Subsystem Design Matrix

[18] http://www.tacticallanguage.com/tacticaliraqi/

Currently the U.S Army has two active programs researching and developing new technologies to assist the dismounted soldier in the field: “Land Warrior” and the larger “Future Combat Systems” of which Raytheon is the prime sensor integrator. As seen below in Figure 21, the “Land Warrior” concept is a fairly complex and expensive set of equipment. Other systems proposed (such as the Eyekon) include complex, expensive optical based helmet mounted displays (see Figure 22). Although no dollar amount has been published, I would expect the price to be in the thousands of U.S dollars. The SPSD will not fulfill all the requirements of the Land Warrior system, but it will meet many of them for pennies on the dollar.

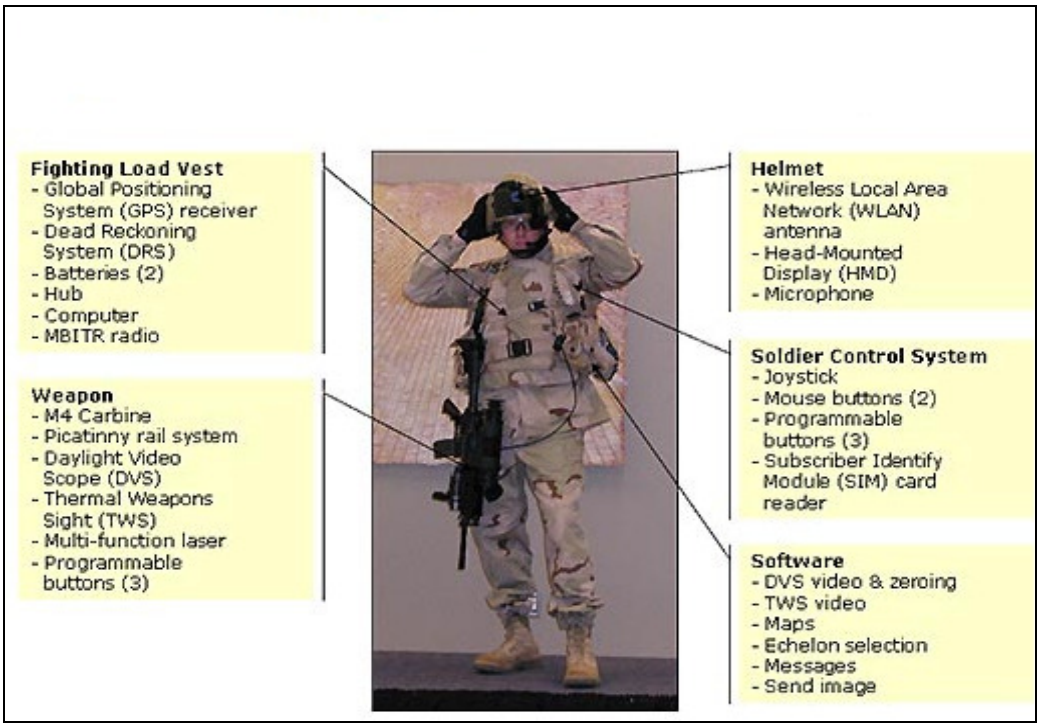

Figure 21: Land Warrior Combat System

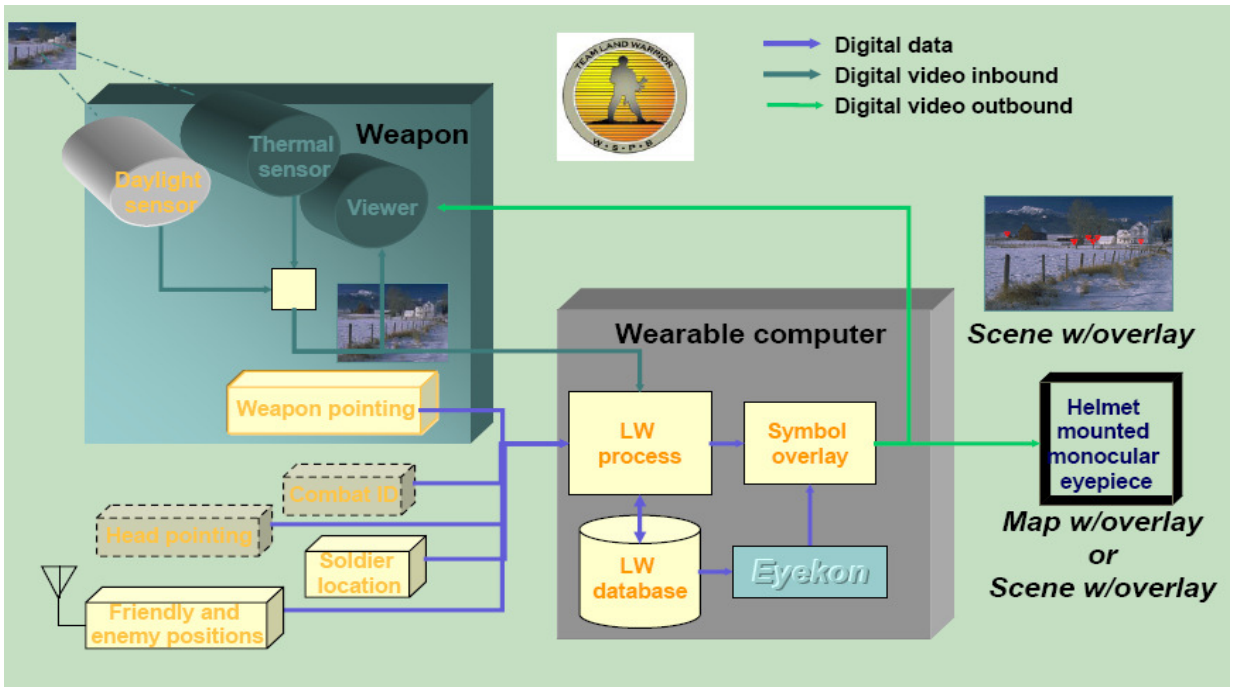

Figure 22: Eyekon System

[21CSI Systems]

In comparison, the Future Combat (Warrior) System (FCW) system contains a Sensors & Communications Vision subsystem for the soldier containing the following: squad level communications using JTRS Cluster 5 Soldier (Software) Radio, state-of-the-art distributed and fused sensors, tactical intelligence and collection assets, **enhanced situational awareness**, embedded training, **on-the-move planning**, and **linkage to other Future Combat System assets**. The SPSD does not intend to replace any of the above systems, only serve as a cheap, interim solution. The SPSD could be built and fielded now with COTS hardware and software; whereas the other proposed systems are years away from fielding.

Figure 23: Future Warrior System (FWS) System [U.S. Army]

4.3.4.1.2.1 Agent Based Expert System Detail

I propose using smart agent and mobile agent technology to enable a soldier with a GPS-driven, graphical hand-help geographic situation display (SPSD) to assign an agent via touchpad to monitor a specific area of interest (AOI), such as areas of previous IED detonations, or areas known to be frequented by enemy combatants. Using (wireless) mobile agents, all SPSDs participating within the same wireless network would exchange "intel-data" used to feed the rule-based software warning monitor/engine. An example might be where a dismounted patrol notices the presence of a particular vehicle or group of individuals known to be potentially hostile. A soldier would input the data into his or her SPSD and the mobile agents would migrate throughout the wireless network monitoring for data *triggers* input from other asset*s*, feeding the AI based local smart agents who would then queue all soldiers within the area via graphic or audible warning of the potential hazard.

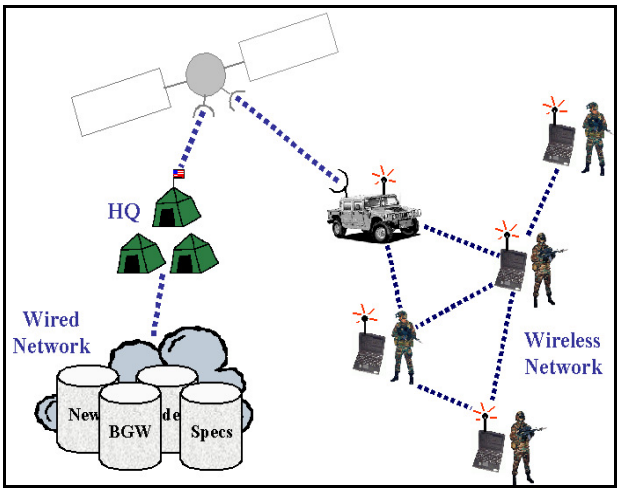

Figure 24: Agent Enhanced Wireless Network

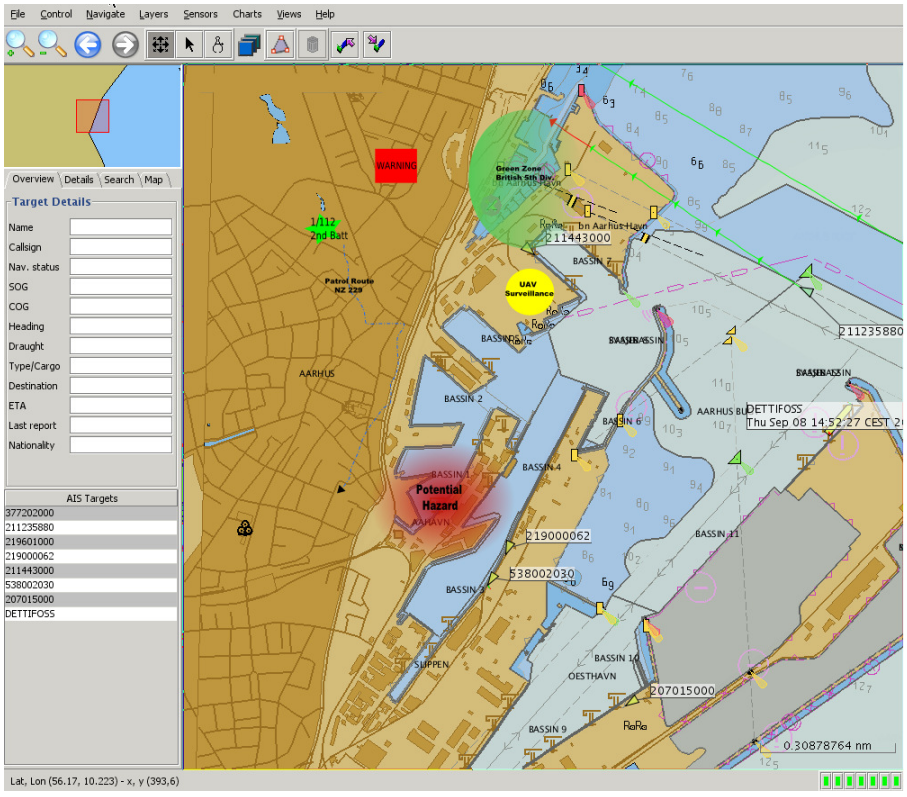

Figure 25: Agent-Based Guidance Display [(modified) Agile Systems]

Such a model as I propose has been described by DARPA in partner with Lockheed-Martin. They conducted research involving the use of agents in military applications which they called *Domain Adaptive Information System* (DAIS) specifically for use in military Command and Control (C^2) systems.

At the highest level – the model can be represented as a sensor-based mesh network. *McGrath, Chacon and Whitebread* [19] chose to represent the topology using three agent entities or operations:

1) Information Pull (i.e. database queries)
2) Information Push – (i.e. report dissemination to small units)
3) Sentinel Monitoring – (i.e. conflict monitoring) implemented with a Java based framework called *Extendable Mobile Agent Architecture* or (EMAA) [20].

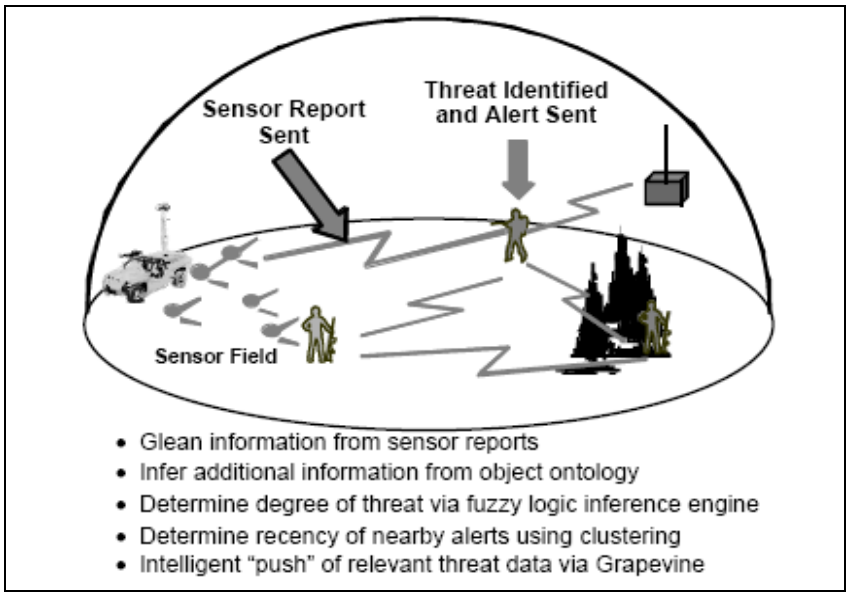

Figure 26: Agent Based Warning System

In the same paper they also describe a "Distributed Event Messaging System" used for agent communications. Likewise, Cougaar provides a rich and deep set of tools such as the "*Message Transport Service"* (MTS), which is an adaptive JVM-level (Java) service that handles all inter-agent

communication within a Cougaar cluster. The MTS is componentized and includes adaptive features that can be selected at runtime. Asynchronous transport protocols can be plugged into the MTS as components, where standard protocols include RMI, CORBA, HTTP, and UDP. Granted this entire agent-based software subset as described does not exist in a COTS/Component package, but Cougaar provides the open source tools needed to design, develop and debug a robust agent based system. *Helsinger, Thome and Wright* [21] detail the services provided by Cougaar, such as persistence, plug-ins, quality of service (QOS), and a yellow-pages (YP) based network lookup service. Note: Cougaar also provides a lightweight embedded version ("MicroEdition") of its agent infrastructure ideally suited for our host processor. Other open source toolkits such as Tryllian [19] exist, but in the author's opinion, Cougaar is better supported and seems to have a wider development community with excellent documentation.

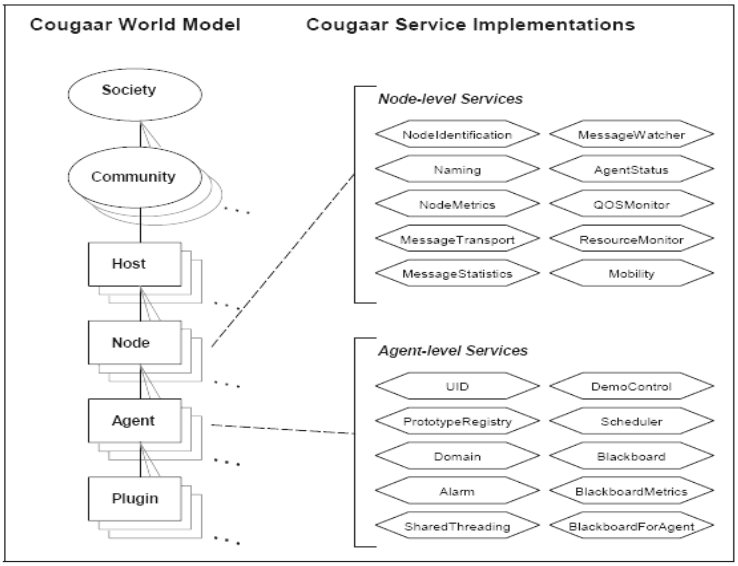

Figure 27: Cougaar Service Hierarchy

[19] http://www.tryllian.org/

4.3.4.1.3 Geo-Orientation Subsystem

The heart of the SPSD is the Geo-Orientation subsystem. Currently fielded military GPS systems such as the Precision Lightweight GPS Receiver or (PLGR) are bulky and have no [20] graphic display, and more importantly - hard to come by. From my experience in the U.S. Army within the Combat Arms community, many soldiers resorted to using COTS based GPS receivers such as the Garmin GPS-12 when in the field, as PLGRs were near impossible to acquire. I also might add that since Selective Availability has been disabled – experience has shown me that my COTS based 12 channel receiver (Garmin GPS-12) is more accurate than the 5 channel PLGR unit although the PLGR has anti-spoofing features. "GPS Spoofing"[21] or feeding of or broadcasting to a GPS receiver false data is mentioned but not covered in this paper as it is beyond the scope of my discussion and intent.

Figure 28: Precision Lightweight GPS Receiver (PLGR) [Rockwell Collins]

[20] [Pathfinder Magazine] "PLGR Only Authorized GPS Receiver for Combat Ops", Jul 2003
[21] Warner and Johnson, "GPS Spoofing Countermeasures", Los Alamos Labs, LAUR-03-6163

Functional Requirements	**Design Parameters**		
	PDA	***GPS H/W***	***GPS Open Source S/W***
Unit to Provide Soldiers with Geographic Location	X	X	
Unit to Display other Soldiers, Points of Interest or Hazards	X	X	X

Table 13: Secondary Geo-location Design Matrix

The SPSD will not only graphically display its current position; it will also display anyone else's position on the map grid that is also equipped with an SPSD. Virtually anything of any tactical value could also be displayed on the SPSD such as other members of a squad or platoon, vehicles, command posts (CPs), etc. It is also a requirement (Design Constraint) that all participants of the Wi-Fi 802.11 communications net will use strong (at least 128 bit) encryption.

The current implementation of the SPSD uses an external GPS receiver connected to the SL-5500D via serial cable. Running onboard the SPSD is an open source component called "**gpsd**[22]**"**, which is a monitor daemon that listens on a selectable TCP/IP port waiting for applications to request information from GPSs or differential-GPS radios attached to the host processor. Each GPS receiver is expected to be direct-connected to the host via a USB or RS232C serial port. At startup, gpsd discovers the correct port speed and protocol to use (NMEA-0183).

Using **gpsd** as a baseline software component, I propose adding the following:

1. Modify the gpsd daemon process to read in the one-per-second (1Hz) NMEA sentence of the format mentioned in section **4.2.5.2, (**NMEA Sentence Format), and concatenate the user id, etc. onto the end of the NMEA sentence (Figure 29), and then broadcast this as an encrypted message

to all participants of the 802.11 network. Each participant (SPSD) would then receive this message and the NMEA sentence would be parsed, reformatted and sent to the local gpsd's TCP/IP socket where it would be processed and displayed on the SPSD screen.

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,**MY_USER_ID**,*47

Figure 29: Modified NMEA sentence

2. Modify the **gpsd** code to read incoming broadcast messages or code another lightweight process to read broadcast messages from remote SPSD units and assign a unique icon to each unique user id to enable target differentiation on the SPSD display.

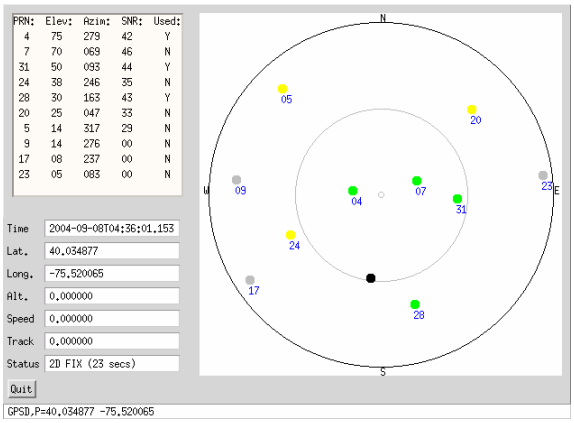

Figure 30: Functioning gpsd display (satellite ephemeris)

[22] http://gpsd.berlios.de/

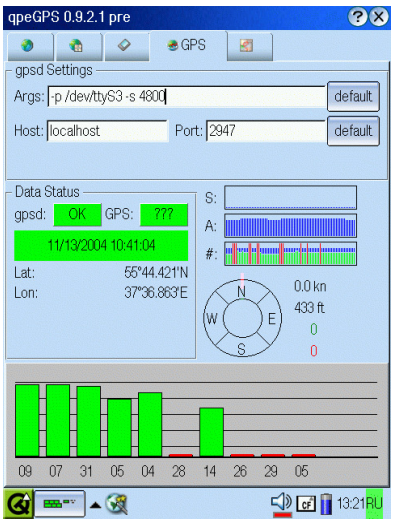

Figure 31: Functional qpeGPS/SPSD display

This design will enable all participants of a broadcast network to see each and everyone else's detailed geographical position with minor code modifications. **Note**: Use of the commercial (COTS) Garmin GPS-12 was borne of necessity (money) as I was limited on funds and already owned a Garmin GPS-12 receiver. This implementation is strictly to demonstrate the feasibility of the SPSD. There are currently many fielded single chip GPS systems [23] on the market and if theoretically fielded, the units could easily be incorporated into a single handheld unit

4.3.4.1.4 GIS Display Mapping

The display of geographic data on computer systems is typically done via Geographic Information System (GIS) software and hardware systems. GIS systems provide facilities for creating, analyzing managing and storing map data all in a standardized format. The agent-based software would

[23] [SiRFstarIII™ GPS Single Chip] http://www.sirf.com/Downloads/Collateral/GSC3(f)_6.20.05.pdf

be required to provide visual queues to the GIS-based software driving the SPSD GPS display. The open-source community provides a rich set of GIS related tools which are 100% compliant with current GIS standards guaranteeing full inter-operability with all GIS hardware and software. After researching the available tools, "*Open-Map*" seemed to provide the features needed. OpenMap is an open source, java-based toolkit for developing applications that manipulate GIS formatted geographic information. OpenMap components can access data from other GIS applications locally or distributed.

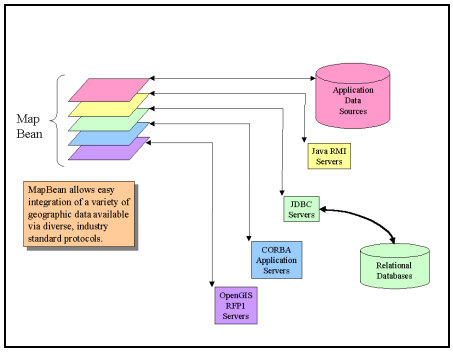

Figure 32: OpenMap Infrastructure [BBN Technology]

Current Army doctrine dictates that GIS-system equipped HMMWVs assist the command and control (C^2) structure by producing current up-to-date digital hard and soft format maps. These maps could be either distributed to the infantrymen on compact-flash or encrypted and transmitted via 802.11 to enable the display of fresh up-to-date topographic information on the SPSDs.


The Digital Topographic Support Systems — Light (DTSS-L). DTSS-L is a Humvee-mounted enclosure that provides 3D map and terrain data as tactical decision aids including visibility/line of sight, mobility, and situation-specific query analyses.

Figure 33: HMMWV-based Mobile Mapping Vehicle [U.S. Army]

4.3.4.1.5 RF Communications Subsystem (Wi-Fi)

While the GPS subsystem of the SPSD may be the heart and the Agent subsystem its brain, its backbone is its RF wireless capability via 802.11. The requirements for the wireless capability of the SPSD were also quickly synthesized using Axiomatic Design techniques. During requirements generation, Axiomatic Design allowed me to quickly determine I needed either a Compact-Flash (CF) or Secure Digital (SD) solution for a wireless interface. This assisted me in choosing a Sharp SL-5500D as it has **both** CF and SD interfaces. The only PDA with both interfaces I was able to locate. The final constraint on determining the Wi-Fi interface was money ($). Once again as this is a prototype unit – I choose the cheapest solution I could find – a Socket Technologies (Figure 15) 802.11b CF card. Axiomatic Design techniques assisted greatly in this case also, as it allowed me to quickly see that a coupling existed between the operating system versions and the wireless card. I was able then to insure that I could locate a compatible wireless card. The Socket P500 is 100% compatible the open source Linux kernel/device driver (2.4.18+ and 2.6.X) and worked perfect the first time!

The second required piece of hardware to complete a deployable tactical network which supports the SPSD is the Fortress ES520 portable wireless access point (Figure 17). The SPSD was conceived to be deployed in a harsh environment where most likely there will be no available network infrastructure (areas such as Iraq or Afghanistan). To operate an extended, long-range Wi-Fi network, a base station or "Access Point" is required, as a simple Ad-Hoc/Peer-to-Peer network would be totally unusable as the range would be limited to approximately 300 ft. See section 3.6 for introduction to Wi-Fi networks.

Functional Requirements	**Design Parameters**		
	PDA	***Wi-Fi H/W***	***Wi-Fi Open Source S/W***
System to TX/RX GPS Data	X	X	X
System to TX/RX Data	X	X	X

Table 14: Secondary RF Communications Design Matrix

4.3.5 Definition of Modules

It is during this phase of development that a divergence from traditional axiomatic design was seen fit. Suh states at this point "At the leaf level, any software programming language, e.g. C++ or Java can be used to write the modules as long as they are internally consistent …..". Since one of the major objectives of this endeavor is the utilization of COTS based open-source software, the techniques proposed by *Dogru and Tanic* [22], and *Repenning, Ioannidou, Payton, Ye and Roschelle* [23] involving techniques for Component Oriented Software Engineering were adopted.

4.3.5.1 Component Oriented Software Engineering Detail

COSE can be defined as the development and reuse of software elements which offer a predefined service and are able to communicate with other elements or "components". Software component(s) typically share the following traits:

- ✓ Multi-use
- ✓ Non-context-specific
- ✓ Ability to easily integrate with other components
- ✓ Encapsulated or non-investigable through its interfaces
- ✓ Singular unit of independent deployment and versioning

With this definition, it is easy to realize *open-source software is a rich field for component-based development and re-use harvest.* However, some software will need to be developed (knowledge-based expert system) since no current open-source project fully meets all the requirements. As noted previously, there are many open-source subcomponents available to quickly construct a knowledge-based expert

system or “super-component” [24]. *Dogru* and *Tanik* refer to this paradigm as a “component-oriented” approach as opposed to a “component-based” model, where “component-oriented approach emphasizes the composition of existing components”.

As the development of the SPSD is by definition a very short period development process, some form of rapid-prototyping model was needed. *Repenning, Ioannidou, Payton, Ye and Roschelle* propose a model called “CORD” or Component-Oriented Rapid Development [9]. The CORD model was investigated but was found to be too heavy-weight and more “team” oriented. CORD proposes a number of parallel threads or distributed discrete teams working on sets of components. This was determined to be unusable as the development team was composed of just one developer - myself. Hence a lighter weight model was needed. I searched and found a development model that met my requirements, adopted itself elegantly to the COSE model, and would enable me to deliver the SPSD on-time. This development model being “*Test-Driven Development*”[24].

[24] Dogru, Tanik, “Process Model for COSE” IEEE Software 2003

4.3.5.2 Test-Driven Approach for COSE

"Test-Driven" (TD) Development involves iteratively writing a test case and then implementing only the code necessary to pass that singular test only. Several features immediately leap out at the developer seeking to either use or reuse software components:

1) the delivery of rapid feedback related to the application to the developer when using TD
2) the Test-Driven model can be applied to the improvement of existing software and removal of software defects from legacy code that was not developed using TD [24] as described in text.

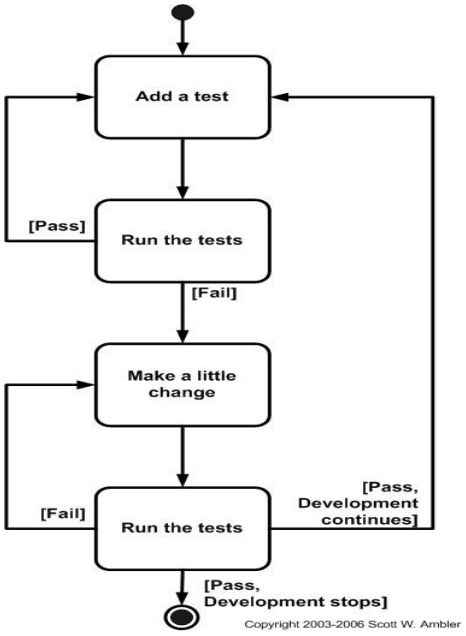

Figure 34: Test-Driven Development Cycle

Kent Beck, the creator of "eXtreme Programming" (XP) defines two simple rules for Test Driven Development [24]. One, "you should write new code only when an automated test has failed". Second, "you should eliminate any duplication that you find". Beck explains in his text [24] how these two simple rules drive the TD development process:

1) "You design organically, with the running code providing feedback between decisions."
2) "You write your own tests because you can't wait 20 times per day for someone else to write them for you."
3) "Your development environment must provide rapid response to small changes (e.g. you need a fast compiler and regression test suite)."
4) "***Your designs must consist of highly cohesive, loosely coupled components*** (e.g. your design is highly normalized) to make testing easier (this also makes evolution and maintenance of your system easier too)."

It is of the author's opinion that this particular development model fits exceptionally well with the COSE model because a.) By definition – the components must already exist per the definition of a "component" as defined earlier, and b.) Point 4 defined by Beck – components are also by definition, highly cohesive and loosely coupled. I propose to use Axiomatic Design to provide the input for Test-Driven development by adding the *additional* first 2 steps shown below.

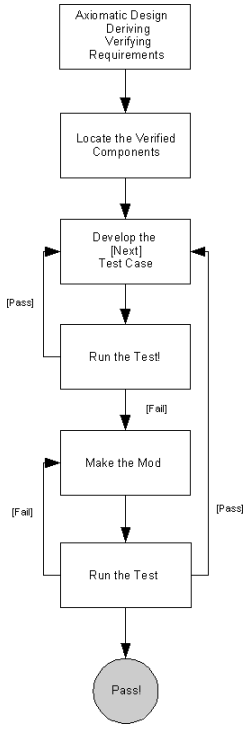

Figure 35: Proposed Axiomatic Design *feeding* Test-Driven Design

4.3.5.3 Module Detail, Attributes, and Operations

Essentially all required components exist already "assembled", except for the Knowledge-Based "Artificial Intelligence" system used to assist the dismounted soldier with the tactical decision making process.

4.3.5.4 Establishment of Interfaces

The interfaces between all of the COTS based components are predefined and were able to be used as-is with little or no modification. The GPS module(s) (gpsd and qpeGPS) were already designed to communicate via TCP/IP socket. The modifications needed to interface to the wireless module would be expected to listen on the same socket; hence virtually no interface changes are needed.

The Agent based software would need to be developed using an open source toolkit. After a side by side comparison, I elected to specify "Cougaar", as it has a rich and well supported java-based infrastructure that support both massively distributed and tiny, embedded development. The agent-based architecture I envisioned would be distributed but not massively so ("where bigger is better"). Being a distributed system, I would propose that a rapid-prototyped development cycle leading to deployment be adopted. That is, where one infers "throw-away" or "one-off" intent, where the application would be discarded using standard "rapid-prototyping" – I would use the "Test-Driven" (TD) approach of continuously refining the prototype. *Togay, Dogru, Tanik and Grimes* [18] discuss such a methodology ("Component Oriented Simulation Development with Axiomatic Design") and use "High Level Architecture" (HLA). The authors [18] use HLA as an example/model as HLA adheres to OMT standards for interface design.

High-Level Architecture or (HLA) is a DOD standard for inter-connecting computer simulation

systems enabling them to run together and exchange information. As opposed to building large monolithic systems from the ground up, High-Level Architecture provides the standards allowing you to combine preexisting simulation-based systems with newer systems. HLA provides the ability to reuse preexisting systems for new development. HLA also provides for using diverse programming languages and/or operating systems. The High Level Architecture (HLA) provides the following components:

- Interface Specification. The interface specification document defines how HLA compliant simulators interact
- Runtime infrastructure (RTI). (***Cougaar***) The RTI provides a programming library and an application programming interface (API) compliant to the interface specification.
- Object Model Template (OMT). The OMT specifies what information is communicated between simulations and how it is documented.
- HLA Rules. Rules that simulations must obey to be compliant to the standard.

When implementing an HLA-based system, an HLA compliant simulation is referred to as a *federate***.** A *federation* is defined as a set of simulations connected via RTI using a common OMT. An *object* is a set of related data containing attributes or data-fields, sent between simulations. Interactions are defined as events sent between simulations and may also contain parameters or data fields.

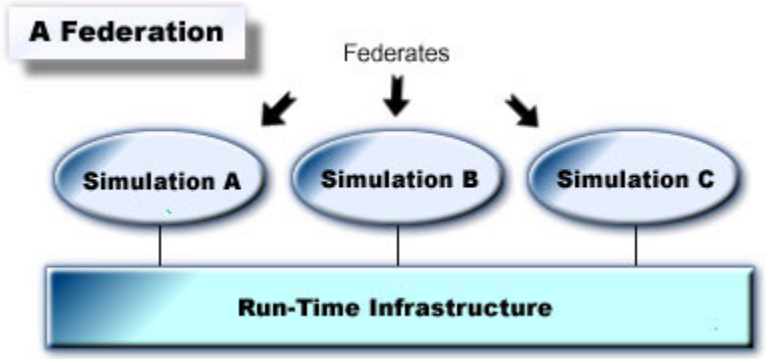

Figure 36: High-Level Architecture Environment [Pitch HLA]

4.4 GPS Statistical Accuracy Predictions

Accuracy and precision are two terms used to describe and quantify how "good" or accurate the acquired position of a GPS receiver is. Accuracy is the degree of closeness of a prediction related to its true value. Precision is the degree of closeness of all observations to their means. As shown in Figure 37 below, the true position is located in the intersection of the cross hairs. The centroid of the shaded area is the mean estimated position, while the radius is the measured uncertainty.

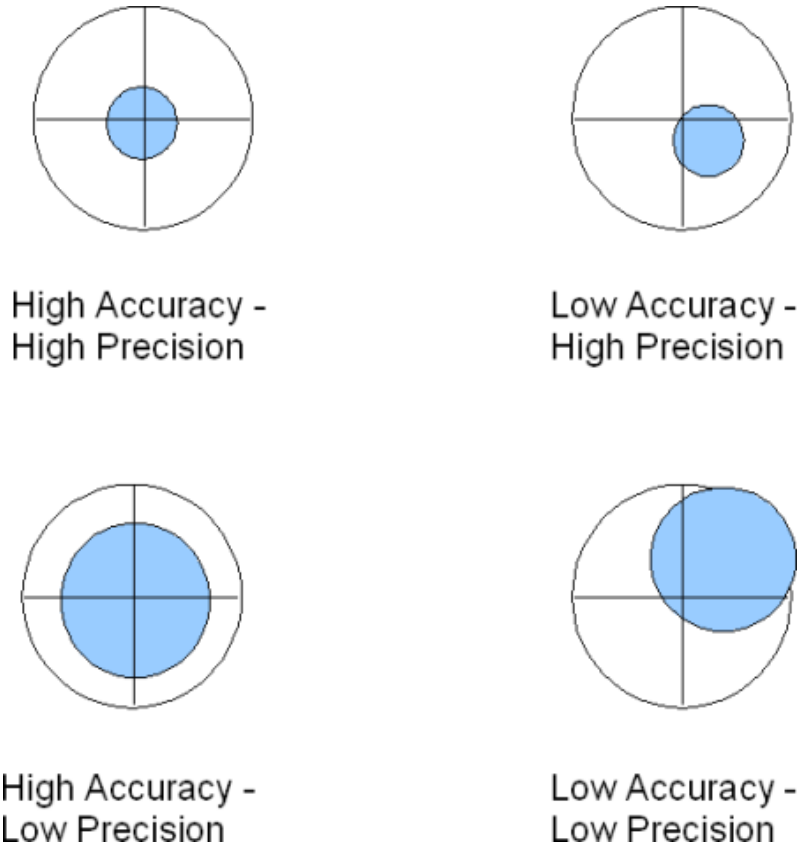

Figure 37: GPS Accuracy Correlations

As an example, if you know the current position given as *Latitude(a), Longitude(a)*, and the location that a GPS receiver reports to you is *Lat(b), Lon(b),* you can compute the current error with:

$$Lat_{err}, Lon_{err} = Lat_b, Lon_b - Lat_a, Lon_a$$

If you take a GPS navigation fix elsewhere in the field at *Lat(c), Lon(c),* you can compute an accurate position, by adding in the offset:

$$Lat_{good}, Lon_{good} = Lat_c, Lon_c - Lat_{err}, Lon_{err}$$

The US Coast Guard uses this approach near coastal waterways and transmits the datum via long wave radio stations and their system is free to use. In many areas, differential GPS may be available via FM broadcast station typically on a fee based schedule.

Note: On May 1, 2000, Selective Availability or (SA), which resulted in the intentional degradation of GPS signals, was terminated. With this enhancement, a GPS without a Differential GPS unit became almost as accurate as a GPS with a Differential GPS.

4.4.1 Calculating Horizontal Position Accuracy

Calculating the horizontal accuracy of a GPS receiver involves determining the degree of conformance between the estimated or measured position, time, and/or velocity of a GPS receiver and its true time, position, and/or velocity as compared with a constant standard. Radio navigation system accuracy is usually presented as a statistical measure of system error and is characterized as follows:

- **Predictability** - The accuracy of a radio navigation system's position solution with respect to the charted solution. Both the position solution and the chart must be based upon the same geodetic datum.

- **Repeatability** - The accuracy with which a user can return to a position whose coordinates have been measured at a previous time with the same navigation system.
- **Relativity** - The accuracy with which a user can measure position relative to that of another user of the same navigation system at the same time.

To calculate the two dimensional position errors, the Distance Root Mean Squared (DRMS) is used where:

$$\textit{Distance\_Root\_Mean\_Squared} = \sqrt{\sigma_x^{\,2} + \sigma_y^{\,2}}$$

DRMS is a scalar number that expresses 2D accuracy. In order to compute the DRMS of a horizontal errors, the standard errors (σ) from the known position(s) in the direction of the Cartesian coordinate axis (X, Y) are required.

The **Standard Deviation** (σ) or the square root of the variance, measures how spread out the values in a data set are. If the data points are all similar, then the standard deviation will be low (closer to zero). If the data points are highly variable, then the standard variation is high (further from zero). The standard deviation is always a positive number and is always measured in the same units as the original data. For example, if the data are distance measurements in meters, the standard deviation will also be measured in meters. Measured position errors of GPS receivers can range from tens of meters for recreational uses to a few millimeters for the survey of land, depending on equipment, signals and usage.

Another method of calculating and representing GPS horizontal error is **Circle Error Probability** or (CEP). CEP refers to the radius of a circle in which 50% of the values occur, i.e. if a CEP of 5 meters is quoted then 50% of the horizontal point positions should fall within 5 meters of the true position. The circle error of probability (CEP) is formally defined as the size of the circle that encompasses 50% of all the location fixes. So, if you collect 100 position fixes and select the 50 closest

ones, the circle that includes these is how certain you can be of your location. Simply stated, a CEP of 100 meters means that 50% of all position fixes are within 100 meters.

$$\textbf{\textit{Circle\_Error\_Probability}} = 0.62\sigma_y + 0.56\sigma_x$$

Similar to the CEP, the **Distance Root Mean Square** (DRMS) describes 67% of all position fixes, 2 DRMS describes 95% of the fixes, and 3 DRMS describes 100% of the fixes. Figure 38 illustrates this pattern.

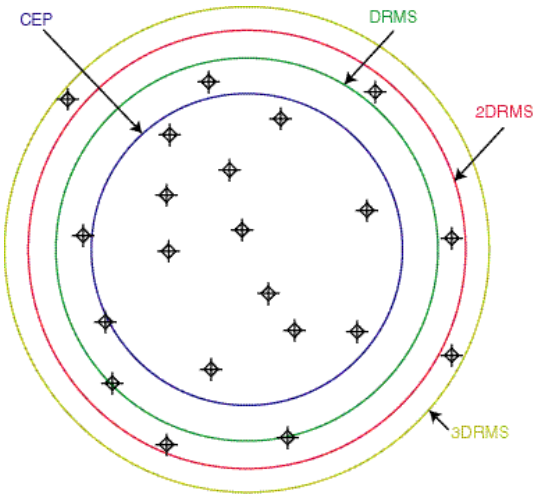

Figure 38: Circle Error Probability/Distance Root Mean Square Error Scatter Plot

4.4.2 GPS Accuracy Conclusions (Is it good enough?)

As mentioned earlier, the intentional degradation of GPS signals known as *Selective Availability* was **terminated** as of May 2000. With this enhancement, a GPS without a Differential GPS unit became almost as accurate as a GPS with a Differential GPS attached. As observed in the Garmin factory specifications (Table 6), with Selective Availability enabled the predicted accuracy of the Garmin GPS 12 is 100m (2DRMS). The Garmin factory specification also advertises a predicted accuracy of (1-5) meters using a Differential GPS beacon.

As observed from the collected statistical sampling and testing (Appendix 7.2), my calculated accuracies were as follows: 3.52m 2DRMS, 3.399m 2DRMS, 2.620m 2DRMS and 2.923m 2DRMS. The mean of the 4 sample DRMSs is 3.091 meters. These results compare quite favorable with Garmin's published accuracy data. Since two of the four sample histories were taken during intense rain and the results compared favorably with those taken during clear atmospheric conditions, it can be surmised that signal attenuation due to rainfall is negligible. (Note: before this could be taken as fact more strenuous testing would be needed. Although information gleaned from internet searches states that rainfall attenuation of GPS signaling is negligible).

In closing, it is of the author's opinion that a 1 to 3 meter predicted horizontal GPS accuracy is more than adequate for the intended use providing the dismounted soldier with a device capable of tracking and marking objects of a tactical nature, and providing a grid-based situational display with a 5 to 10 meter scale.

4.5 802.1X Wi-Fi vs. WiMAX (Statistical Range Predictions)

Typical outdoor ranges for 802.11g range from 1800 ft to 2500 ft. This would in the authors' opinion, be perfectly suitable for use at the squad level on typical patrol duty. It's also worth noting there are many workarounds available (range extenders/amplifiers). I would propose investigating the emerging new 802.16 standard known as "WiMAX". WiMAX offers exponential increases in range and throughput compared to standard 802.11X Wi-Fi as well as better security features. The limited range, bandwidth and throughput of Wi-Fi limits the dismounted soldier to relative close ranges to access points or peers. On the contrary, a single WiMAX base station can service a wide geographic area (30 to 40 square miles).

Examples of WiMAX 802.16 ranges:

- Hawaii, through dense palm trees and rolling hills (non line-of-site), (1.86 miles), 26 Mbps
- Ireland, through ancient cathedrals and other landscape ruins (non-LOS), (7.45 miles), 24 Mbps
- Quebec City, Canada, partial visibility (optical line of sight (OLOS)), (33 miles), 18 Mbps
- Idaho, USA, (72 miles) with a local mountain partially blocking , 24 Mbps
- Boston, through dense trees (complete non-LOS), (1.86 miles), 26 Mbps
- Markham, ON, Canada, through a business park (non-LOS), (1.86 miles), 56 Mbps

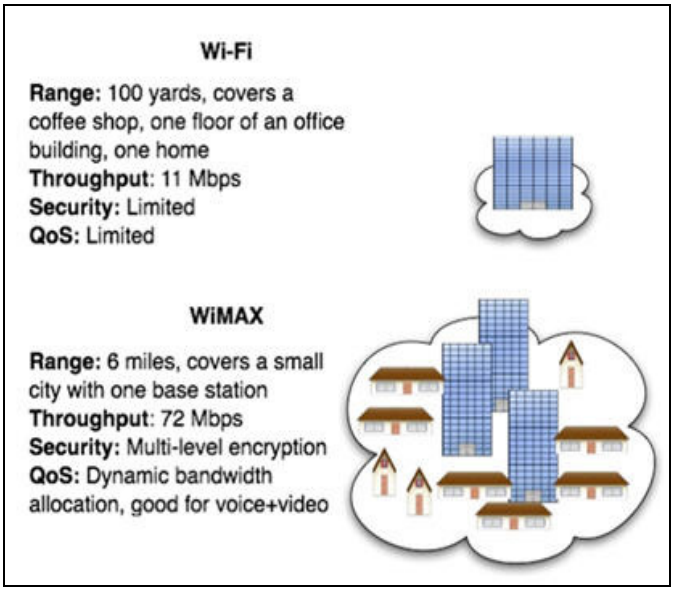

Figure 39: Wi-Fi vs. WiMAX [wimaxx.com]

References

[1] Peters, Look, Quigly, Shrobe, and Gajos, "Hyperglue: Designing High-Level Agent Communications for Distributed Applications", MIT, 2002
[2] Dolev, Gilbert, Schiller, Shvartsman and Welch, "Autonomous Virtual Mobile Nodes", MIT, 2005
[3] Gray, Kotz, Cybenko and Rus, "D'Agents: Security in a multiple-language mobile agent system",

Dartmouth College, 1998
[4] Wendleken, McGrath, Blike, “Agent Based Casualty Care – Medical Expert for Triage Care” Dartmouth College, 2004
[5] Brewington, Gray, Moizumi, Kotz, Cybenko, Rud, “Mobile Agents in distributed information retrieval”, Dartmouth College, 1999
[6] Dogru and Tanik, “A Process Model for Component-Oriented Software Engineering”, IEEE Software, 2003
[7] Dogru, “Component Oriented Software Engineering”, theAtlas Press, 2006
[8] Suh and Doh, “Axiomatic Design of Software Systems” Axiomatic Design Solutions, Inc. MIT
[9] Repenning, Ioannidou, and Payton, “Using Components for Rapid Distributed Software Development”, IEEE Software, 2001
[10] Jacob, “Not Your Father’s CORBA”, Embedded Systems Journal, 2004
[11] Martin, Cheyer, and Moran, “The Open Agent Architecture: A Framework for Building Distributed Systems”, SRI Intl, 2004
[12] Mitre , “Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense”, Mitre Corp., 2003
[13] Ackerman, “Army Teaches Soldiers New Intelligence-Gathering Role”, Signal Journal, April 2005
[14] FM Field Manual Operational Terms and Graphics United States Army United States Marine Corp
[15] Suh, Axiomatic Design: Advances and Applications, Oxford University Press, Ch 4, 2001
[16] Suh, Axiomatic Design: Advances and Applications, Oxford University Press, Ch 5, 2001
[17] Dogru, Tanik, and Grimes, “Component Oriented Simulation with Axiomatic Design”, IDPT-2006
[18] Losh, "In Country with Tactical Iraqi: Trust, Identity, and Language Learning in a Military Video Game”, UC Irvine, 2005
[19] McGrath, Chacon and Whitebread, “Intelligent Mobile Agents in Military Command and Control”, Proceedings of the Autonomous Agents Conf., 2000
[20] Letini, Rao Theis, Kay, “EMAA: An Extendable Mobile Agent Architecture”, Fifteenth Natl. Conf. on AI, 1998
[21] Helsinger, Thome, Wright, “Cougaar: A Scalable Distributed Multi-Agent Architecture”, IEEE SW 2004
[22] Dogru, Tanic, “A Process Model for Component-Oriented Software Engineering”, IEEE Software 2003
[23] Repenning, Ioannidou, Payton, Ye and Roschelle, “Using Components for Rapid Distributed Software Development”, IEEE Software 2001
[24] Beck, “Test Driven Development: By Example”, Addison Wesley, 2003