2013-09

Enhancing entity level knowledge representation and environmental sensing in COMBATXXI using unmanned aircraft systems

Teters, James C.,II

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ENHANCING ENTITY LEVEL KNOWLEDGE REPRESENTATION AND ENVIRONMENTAL SENSING IN COMBATXXI USING UNMANNED AIRCRAFT SYSTEMS

by

James C. Teters II

September 2013

Thesis Advisor:	Imre Balogh
Second Reader:	Peter Nesbitt

This thesis was performed at the MOVES Institute
Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2013	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Enhancing Entity Level Knowledge Representation And Environmental Sensing In COMBATXXI Using Unmanned Aircraft Systems			**5. FUNDING NUMBERS**
6. AUTHOR(S) Teters, James C. II			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** N/A
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number N/A.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Current modeling and simulation techniques may not adequately represent military operations using unmanned aircraft systems (UAS). A method to represent these conditions in a combat model can offer insight to the use and application of UAS operations, as well as understanding the sensitivity of simulation outcomes to the variability of UAS performance. Additionally, using combat model simulations that do not represent UAS behavior and conditions that cause this variability may return misleading or incomplete results. Current approaches include explicit scripting of behaviors and events. We develop a proof of principle search, targeting, and acquisition (STA) model for use with UAS within COMBATXXI, leveraging existing STA research conducted at the MOVES Institute at the Naval Postgraduate School. These dynamic behaviors are driven by events as they unfold during the simulation run rather than relying on preplanned events as in the scripted approach. This allows these behaviors to be highly reusable since they do not contain scenario or incident specific information. We demonstrate the application of the new STA model in a tactical convoy scenario in COMBATXXI. A design of experiments and post analysis quantifies the sensitivity of the measures of effectiveness of success to conditions contributing to variability in UAS performance.

14. SUBJECT TERMS UAS, UAVs, Unmanned Aircraft Systems, Unmanned Aerial Vehicle, Sensors, Electronics & Electronic Warfare, Information Systems Technology, COMBATXXI, HTNs, Hierarchical Task Networks, Ontology, Reasoning, Knowledge Representation, Training			**15. NUMBER OF PAGES** 145
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ENHANCING ENTITY LEVEL KNOWLEDGE REPRESENTATION AND ENVIRONMENTAL SENSING IN COMBATXXI USING UNMANNED AIRCRAFT SYSTEMS

James C. Teters II
Major, United States Army
B.A., Marshall University, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN
MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
September 2013

Author: James C. Teters II

Approved by: Imre Balogh
Thesis Advisor

Peter Nesbitt
Second Reader

Christian Darken
Chair, MOVES Academic Committee

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current modeling and simulation techniques may not adequately represent military operations using unmanned aircraft systems (UAS). A method to represent these conditions in a combat model can offer insight to the use and application of UAS operations, as well as understanding the sensitivity of simulation outcomes to the variability of UAS performance. Additionally, using combat model simulations that do not represent UAS behavior and conditions that cause this variability may return misleading or incomplete results. Current approaches include explicit scripting of behaviors and events. We develop a proof of principle search, targeting, and acquisition (STA) model for use with UAS within COMBATXXI, leveraging existing STA research conducted at the MOVES Institute at the Naval Postgraduate School. These dynamic behaviors are driven by events as they unfold during the simulation run rather than relying on preplanned events as in the scripted approach. This allows these behaviors to be highly reusable since they do not contain scenario or incident specific information. We demonstrate the application of the new STA model in a tactical convoy scenario in COMBATXXI. A design of experiments and post analysis quantifies the sensitivity of the measures of effectiveness of success to conditions contributing to variability in UAS performance.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AGL	Above Ground Level
AI	Artificial Intelligence
AK-47	Avtomat Kalashnikova
AT	Anti-Tank
AO	Area of Operations
AOI	Area of Interest
AMSAA	Army Materiel Systems Analysis Activity
BDE	Brigade
BSL	Behavior Specification Language
CO	Company
COE	Center of Excellence
COMBATXXI	Combined Arms Analysis Tool for the 21st Century
COP	Command Outpost, Common Operating Picture
DFP	Deployable Force Protection
DOE	Design of Experiments
DoD	Department of Defense
EOD	Explosive Ordnance Disposal
EO	Electro-Optics
FT	Feet
FOB	Forward Operating Base
FOL	First-order logic
FOR	Field of Regard
FOV	Field of View
GCS	Ground Control Station
HMG	Heavy Machine Gun
HMMWV	High-Mobility Multipurpose Wheeled Vehicle
HQ	Headquarters
HTN	Hierarchical Task Network
KB	Knowledge Base
KPH	Kilometers Per Hour
KR	Knowledge Representation
IED	Improvised Explosive Device
IN	Infantry
IR	Infrared
IOT	In Order To
ISO	In Support Of
ISR	Intelligence, Surveillance, Reconnaissance
LNA	Local National Army
MCCDC	Marine Corps Combat Development Command

M	Meters
MM	Millimeters
MOVES	Modeling, Virtual Environments and Simulation
MSR	Main Supply Route
M&S	Modeling and Simulation
NAI	Named Area of Interest
NPS	Naval Postgraduate School
NVESD	Night Vision and Electronic Sensors Directorate
PBIED	Person-Borne IED
PLT	Platoon
OEF	Operation Enduring Freedom
OP	Observation Post
OWL	Web Ontology Language
RIP	Relief-In-Place
RPG	Rocket-Propelled Grenade
RPK	Ruchnoy Pulemyot Kalashnikova
SA	Small Arms
SPG	SPG-9 Kopye
SOCOM	United States Special Operations Command
STA	Search and Target Acquisition
SWA	Southwest Asia
TTPM	Targeting Task Performance Metric
TRAC	United States Army TRADOC Analysis Center
TRADOC	United States Army Training and Doctrine Command
UA	Unmanned Aircraft
UAS	Unmanned Aircraft Systems
UAV	Unmanned Aerial Vehicle
USAF	United States Air Force
USG	United States Government
UXO	Unexploded Ordnance
WFF	Well Formed Formula
WSMR	White Sands Missile Range
WWII	World War II
W3WC	World Wide Web Consortium

Executive Summary

Current modeling and simulation techniques may not adequately represent military operations in respect to unmanned aircraft systems (UAS), to include the unmanned aerial vehicle (UAV) and its human operator. The performance of a UAS in the field can vary greatly, and this performance can have an enormous affect on the outcome of a military operation. This complex and variable behavior may be sensitive to considerations not represented in contemporary combat models. These conditions can include, but are not limited to, pilot training and experience. Combat models may require a means to model the effects of this behavior, particularly when analyzing the skill or efficiency of a UAS in providing information to other entities in the scenario.

A method to represent these conditions in a combat model can offer insight to the use and application of UAS in operations, as well as understanding the sensitivity of simulation outcomes to the variability of UAS performance. Additionally, using combat model simulations that do not represent UAS behavior and conditions that cause this variability may return misleading or incomplete results. Understanding when and how to model these conditions contributes to quality simulation and analysis.

Current approaches include explicit scripting of behaviors and events. Combat models, including Combined Arms Analysis Tool for the 21st Century (COMBATXXI) [1], provide opportunities for analysis. The ACQUIRE model is the most prevalent in U.S. Army simulations, allowing for the modeling of sensors used for search and targeting and acquisition (STA) [2]. Hierarchical Task Networks (HTNs) allow for automated planning based on changing environments in the simulated world [3]. Ontologies can show the effects that detailed information may play on decision making when utilizing a UAS. This thesis proposes using dynamic behaviors to include hierarchal task networks, ontologies, and sensor models to represent the inherent complexity of a UAS in a simulation. We developed a proof of principle search, targeting and acquisition (STA) model for use with UAS within COMBATXXI leveraging existing STA research conducted at the Modeling, Virtual Environments and Simulation Institute (MOVES) at the Naval Postgraduate School. We showed that this methodology can result in analysis that gives insight into the effects the capabilities of a UAS offer regarding deployable force protection (DFP). An ontology provides a means of representing pieces of information that, when combined with reason, was able to output relevant knowledge of the world and affect the out-

come. This study used a scenario where understanding the ability to identify a target has a relationship to successful execution.

We demonstrated application of the new STA model in a tactical convoy scenario in COMBATXXI. The ability to model a UAS in a working environment is fundamental to a study concerning tactical convoy operations. A design of experiments and post-execution analysis quantified the sensitivity of the measures of effectiveness of success to conditions contributing to variability in UAS performance. Future work is given to identify possible studies that can extend this research.

Acknowledgements

I want to thank Dr. Imre Balogh for his tireless efforts in assisting me with the programming and instruction on HTNs, COMBATXXI, and for making this daunting task not as stressful as it could have been. Also I would like to thank Major Peter Nesbitt, whose help, insight and encouragement have been a tremendous blessing. To Curtis Blais, "Thank you, sir" for your assistance and instruction on building ontologies with Protégé and providing additional eyes on the thesis. Many thanks to the wonderful people at TRAC-Monterey: Lieutenant Colonel John Alt, Major Ed Masotti, and Dr. Tom Anderson for your assistance and a place to work. I also want to thank the awesome faculty at The MOVES Institute for providing a great and wonderful learning experience. It has been a blast.

Thank you Mom, Dad, Nanny and Papaw for believing in me, I hope to never let you down.

Most of all thank you, Bobbi, for your patience, understanding and most importantly, the love you have poured out for me during the last two years; you are the best woman God could have ever given me to call my wife. Thanks Tyler and Emma for being great kids and making me smile at the most perfect times. I love you both.

This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
INTRODUCTION

1.1 BACKGROUND

The United States Military is considered the most powerful and effective fighting force when fighting a traditional red on blue, symmetrical war. Since the September 11, 2001, attacks, our enemies have been successful in mitigating our fighting strength through the use of improvised explosive devices (IEDs). The ability to counter such threats demand an adaptable and resilient force. Modeling and simulation (M&S) plays a vital role in developing scenarios that help leaders make decisions that can have lasting impacts on the battlefield. It is key that M&S provides decision makers with more fidelity and plausible results when studying the potential effects of systems used in theater.

1.2 RESEARCH PROBLEM

Current modeling techniques do not adequately provide the means to modify and measure the effects of changing the properties inherent to human/machine components and processes in an unmanned aircraft system (UAS). This research proposes an innovative way to represent the value of information and training in a constructive simulation, COMBATXXI (Combined Arms Analysis Tool for the 21st Century). Sensors in COMBATXXI do not allow for accurate modeling of IED detection when deploying UAS in a doctrinally correct manner. This may result in improper system deployment, overconfidence of capabilities, or unrealistic expectations causing more harm than good to our Soldiers. This study will attempt to model decision making capabilities of a UAS in COMBATXXI, utilizing newly developed sensor modeling, dynamic planning methods, and ontology based reasoning.

1.3 MOTIVATION

The ability of a unit to effectively perform its mission is dependent on many factors including presence, posture, and safety. The U.S. Army relies on force protection at all times. Unmanned aerial vehicles (UAVs) are gaining in prevalence use since the War on Terrorism began after the tragic events of 9-11. As of 2010, Unmanned Aircraft Systems (UAS) are in use at echelons below battalion and below [5, p. 21]. Analyzing the effective use of a UAS can be performed using the U.S. Army's high-resolution simulation COMBATXXI. This simulation models scenarios ranging from squad level building clearing operations to brigade-size amphibious assaults [1].

COMBATXXI is extremely important for analysis of systems and can be improved to represent the dynamics of a UAS's information-gathering and its effects. Understanding how information and training affect decision making may provide better insight on how we train, equip; and employ systems such as a UAS based on this new prototype. The study utilizes a new means of creating behaviors in COMBATXXI by way of Hierarchical Task Networks (HTNs).

1.4 OBJECTIVES

In order to develop a means to represent information and study the effects of UAS operator skills when using this operation, we identify the following objectives:

- Utilize HTNs to assist in the formulation of behaviors more representative of complex systems in operation.
- Describe a new method to represent knowledge in COMBATXXI through use of an ontology and a new nontraditional sensor model.
- Gather more knowledge on the effects of information fidelity and varying capabilities of sensors utilization of Unmanned Aircraft Systems in a deployable force protection scenario.

1.5 THESIS ORGANIZATION

Chapter 1 lays out the plan to completion. The motivation for the thesis is given and provides the reader with guidance in the direction the study will take in order to answer the research questions.

Chapter 2 provides a review of the UASs that the U.S. Army has in operation and other systems used elsewhere in the Department of Defense (DoD). The development of target acquisition models used in simulations is discussed. Hierarchical task networks are described to provide the reader with an understanding of the value they provide to the developer when compared to previous methods. A brief history of knowledge representation (KR) and its applications are discussed as well.

Chapter 3 details the methodology used in the thesis from its inception to completion of the study. Novel ideas of using HTNs and an ontology approach are discussed, providing the reader with a new method in KR implemented simulations.

Chapter 4 describes the modeling of the Raven (unmanned aircraft of the Raven UAS) using a new STA model. This chapter discusses how an ontology of an IED could allow an entity to make a “realistic” determination of a threat based on the knowledge it is given through the sensor provided by the new STA model.

Chapter 5 provides overall conclusion of the research and a discussion of possible further work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2: LITERATURE REVIEW

2.1 U.S. MILITARY UNMANNED AIRCRAFT SYSTEMS IN OPERATION AND SIMULATION

This chapter provides a review of the UAS in use by the U.S. Military. This includes systems that the U.S. Army has in operation and other systems only being used by other members of the U.S. Armed Forces. A short history of IEDs, their makeup, and means of detection will be examined along with the development of target acquisition models used in simulations. Hierarchical task networks are described in detail to provide the reader with an understanding of the value they provide to the developer when compared to previous methods. A brief history of knowledge representation and its applications are discussed as well.

2.2 UNMANNED AIRCRAFT SYSTEM

The term UAV (unmanned aerial vehicle) is the term most popularly used when discussing an unmanned aircraft that flies with a remotely located pilot. The UAV is part of a system-of-systems called an unmanned aircraft system. According to the U.S. Army [5, p8] "a UAS is comprised of an unmanned aircraft (UA), payload, human operator, control element, display, communication architecture, life cycle logistics, and the supported Soldier." A UAS is capable of bringing the Warfighter value added service with systems such as electro-optical (EO) and infrared (IR) sensors to enhance intelligence, surveillance, and reconnaissance (ISR) capability. According to the U.S. Army 2010 UAS Roadmap, the five different groups of UASs being used are based on the following UA attributes: weight, altitude, and airspeed, see Figure 2.1 [5, p. 12]. These attributes were approved by the Joint Chiefs of Staff on November 25, 2008 [5, p. 12].

2.2.1 Raven RQ-11B

Ravens offer ISR capabilities for units at brigade (BDE) and lower echelons and currently sees high usage in oversea deployments. A report provided to Congress in 2012 reports that the U.S. Army, Navy and United States Special Operations Command (SOCOM) has 5,346 RQ-11 vehicles in the inventory [6, p. 8]. The characteristics of this UAS are given in Figure 2.2. This hand-launched UAV flies low and slow with cruising speeds between 17–44 knots (Figure 2.3). It scans areas of interest (AOI) at an operating altitude of approximately 100–500 feet (30–152

UAS Category	Max Gross Takeoff Weight	Normal Operating Altitude (Ft)	Airspeed	Current Army UAS in Operation
Group 1	< 20 pounds	< 1200 above ground level (AGL)	<100 Knots	RQ-11B Raven
Group 2	21-55 pounds	< 3500 AGL	<250 Knots	No current system
Group 3	< 1320 pounds	<18,000 mean sea level (MSL)		RQ-7B Shadow
Group 4	> 1320 pounds		Any Airspeed	MQ-5B, MQ-1C
Group 5		> 18,000 MSL		No current system

Figure 2.1: Categories of Unmanned Aircraft Systems [5, p. 12]. The U.S. Army UAS categorization is from the "DoD 2009-2034 Unmanned Systems Integrated Roadmap" [5, p. 3]. As of 2010 the Army does not have a Group 2 UAS.

meters) above ground level (AGL). The sensor package allows the operator to gather data via EO or IR cameras using a continuous pan with a +10- to -90-degree tilt. "The [Raven] can be operated manually or programmed for autonomous operation" [7]. The Raven would be the ideal asset for ISR capabilities for any squad-size mission (based on its specs) such as searching for IEDs along a route at known locations.

2.2.2 Scan Eagle

The Scan Eagle UAS is a Group 2 category UAS that performs missions for the U.S. Air Force and the U.S. Marine Corps (see Figure 2.4). This UAV operates at altitudes up to 16,000 feet AGL (above ground level) and can stay aloft for more than twenty hours at speeds between 48–70 knots. The Scan Eagle payload consists of a high-resolution day/night camera and thermal imager. It catapults into operation with a launch and recovery system called Skyhook. This system requires four personnel (two operators, two maintainers) to run [10].

2.2.3 RQ-7B Shadow

The RQ-7B Shadow UAS belongs to the UAS Group 3 category. This UAV can fly to altitudes of 15,000 feet AGL for nine hours of flight. Cruising speed is 90 knots with loitering capabilities at 65 knots. The unmanned aircraft can carry up to 80 pounds of sensors [11]. Figure 2.5 shows a RQ-7 Shadow catapult from its hydraulic rail launcher.

2.2.4 MQ-1C Gray Eagle

The MQ-1C Gray Eagle (also known as "Warrior") belongs in the Group 4 category. It is the upgrade of the MQ-1 Predator. The "M" and "Q" designation mean "multi-role" and "unmanned," respectively. It has a wingspan of 56 feet and a length of 28 feet. This system can perform ISR

Standard Payloads	Dual Forward and Side-Look EO Camera Nose, Electronic Pan-tilt-zoom with Stabilization, Forward and Side-Look IR Camera Nose (6.5 oz payloads)
Range	10 km
Endurance	60–90 minutes (Rechargeable Battery)
Speed	32–81 km/h, 17–44 knots
Operating Altitude (Typ.)	100–500 ft (30-152 m) AGL, 14,000 ft MSL max launch altitude
Wing Span	4.5 ft (1.4 m)
Length	3.0 ft (0.9 m)
Weight	4.2 lbs (1.9 kg)
GCS	Lightweight, Modular Components, Waterproof Softcase, Optional FalconView Moving Map and Mission Planning Laptop Interface, Digital Video Recorder and Frame Capture
Launch & Recovery Method	Hand-Launched, Deep Stall Landing

Figure 2.2: RQ-11B Specifications from Aerovironment [7]. The RQ-11 can fly "low and slow" traveling only 17 to 44 knots which equates to 30 to 60 mph. It is very lightweight 4.2 pounds, making it easy to employ by hand-launching.

and communications relay missions, or be equipped with four Hellfire missiles. It has the ability to fly to 29,000 feet and stay aloft for 25 hours with a maximum speed of 167 knots, see Figure 2.6 [13].

2.2.5 MQ-9 Reaper

Introduced in 2001, the Reaper primarily performs the role of intelligence collection in support of strike, coordination, and reconnaissance missions [15]. It does not belong in the U.S. Army's inventory but can support army missions. The Reaper is a Group 5 UAS with a 66-foot wingspan and is 36 feet. from nose to tail. Its maximum takeoff weight is 10,500 pounds. The hand-thrown Raven weighs a minuscule 4 pounds comparatively. The Reaper (Figure 2.7) flies at a maximum altitude of 50,000 feet and cruises at approximately 230 miles per hour (200 knots) at ranges up to 1,250 miles. It can fire Hellfire missiles or guided bombs, depending on mission requirements.

2.3 IMPROVISED EXPLOSIVE DEVICES

In this section, we provide a definition of an IED, its components, and means of detection in order to assist in making the IED ontology for the scenario to be used in the study.

Figure 2.3: Marine preparing to hand launch the 4.2lb Raven UAV. From [8] The tripod in the back is part of Aerovironment's Ground Control System (GCS), which "improves situational awareness of the ground, provides operators cue to potential threats, and enhances airborne intelligence, surveillance and reconnaissance (ISR)" [9].

2.3.1 History of IEDs

The history of IEDs goes farther back than the "Global War on Terror." They are a direct ambush weapon extremely effective in killing soft targets such as troops and/or lightly armored vehicles such as the High-Mobility Multipurpose Wheeled Vehicle (HMMWV). The DoD states that, "An IED is a device placed or fabricated in an improvised manner incorporating destructive, lethal, noxious, pyrotechnic, or incendiary chemicals and designed to destroy, incapacitate, harass, or distract. ...[the IED] may incorporate military stores, but is normally devised from nonmilitary components" [17, pII-14]. According to the *Washington Post* [18] as of August 29, 2013, 2,503 out of 6,668 [fatalities] 37.5% were caused by IEDs in support of Operation Iraqi Freedom and Operation Enduring Freedom. If an IED explosion does not kill the entire crew in the vehicle, there is a very strong chance that survivors will likely suffer from tramatic brain injury (TBI), loss of limb, burn trauma, etc. The ability to detect and defeat (making IEDs inoperable once found) is paramount to current and future mission successes.

Figure 2.4: Scan Eagle and launch system. From [10]

Figure 2.5: RQ-7B Shadow taking off. From [12]

Figure 2.6: U.S. Army Gray Eagle in flight. From [14]

Figure 2.7: MQ-9 Reaper in flight with landing gear down. From [16]

2.3.2 Makeup of IEDs

In general, an IED [19, para 6-71] is made up of the following components:

1. Main charge (explosive)
2. Casing (materials around explosive)

3. Initiators (what triggers the explosion)
    (a) Command wired
    (b) Radio controlled
    (c) Victim operated
    (d) Timed

2.3.3 Detection of IEDs

It is crucial that modeling detection of IEDs be performed to make it easier for U.S. Army leadership to understand the most beneficial ways to mitigate their effects when using UASs such as the Raven RQ-11B. The most dangerous way of detecting IEDs visually is from close proximity fewer than 5 meters. At distances of 100 meters or more detection is much safer, but more difficult. The enemy is continuously adapting their means of implementing IEDs, which make detection more difficult. Shepherd states in [20] that IEDs were first buried underground, then placed in and behind objects above ground, and then below ground again. Later, they were found hidden in dead animals by the road, vehicles (vehicle-borne IEDs or VBIEDs), and humans (person-borne or PBIEDs). The enemy then began countering our mitigation by placing the IEDs underneath culverts or burying them underneath the roadway. One can see that winning the IED battle is a zero-sum game that costs millions of dollars, and more importantly, the lives and limbs of our soldiers.

According to Shepherd [20] there are four methods to detecting IEDs:

1. "Observing the [observation post] or triggerman."
2. "Identifying the explosives [IED indicators such as loose soil, wires, containers out of place, etc.] or where they are hidden."
3. "Gathering intelligence from the local population."
4. "Being attacked."

Shepherd [20] says that while leaders can use dismounts in staggered column formation or dismounted with vehicles in traveling overwatch, it is not always practical. Another means of detecting IEDs from a safe distance is through the use of unmanned aerial vehicles (UAVs). He sums up the use of aerial reconnaissance below when detecting IEDs:

> [UAVs] assigned to cavalry troops, as well as scouts in helicopters, can provide successful aerial reconnaissance. Aerial surveillance can move quicker and provide

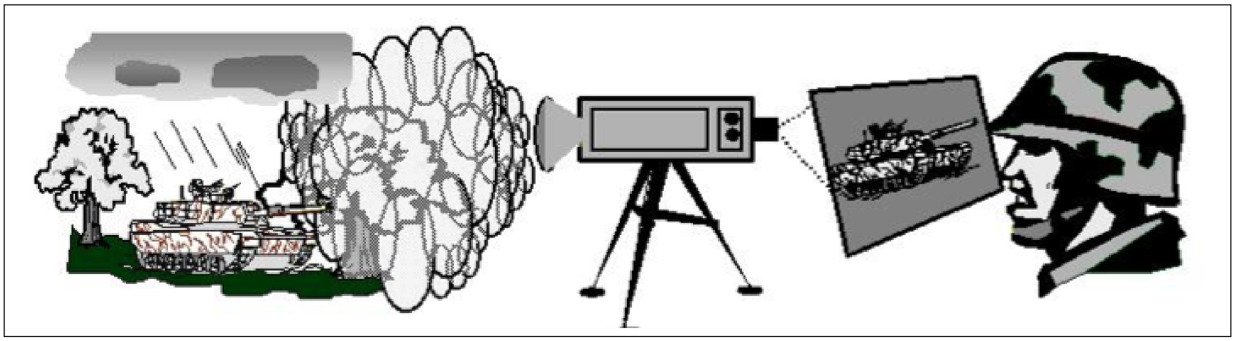

Figure 2.8: Soldier looking through an image device in order to identify an object. From [21, p. 1] This illustrates the point that the M&S community has to take real instruments and tools–like a camera–and create an abstraction (model) the light waves hitting the tank bouncing into the lens of the sensor (here a camera) and then travel back to observer's eyes.

> advanced warning to reconnaissance elements on the ground. Scout elements on the ground can then move forward to confirm or deny information provided by air elements. [20]

This concludes the discussion regarding the makeup and some detection methods for IEDs. Employing UASs to assist in detection of IEDs appears to be useful and promising. Since aerial reconnaissance is important during military operations, it may be necessary to develop methods of modeling this in a simulation. Next, we will look at how human sight has been modeled in the context of target acquisition modeling. This provides a better understanding of the dynamics and limitations of modeling human eyesight in current simulations.

2.4 TARGET ACQUISITION MODELING

In order to engage the enemy, one has to know the enemy's whereabouts. Throughout history, commanders on land or at sea continue to deploy sentries/patrols to gain knowledge of the enemy's location. Since WWII the use of electro-optic (EO) devices have become widespread, and are being utilized ubiquitously throughout the military. Vollermerhausen and Jacobs state: "The history of modeling EO imagers traces back almost 60 years to the pioneering work of Otto Schade... and [he] specifies that an EO image must be produced based on the capabilities and optical characteristics of the eye" [21, p. 9]. Schade had shown that it is possible to model the eye and created an analog model of the eye, which is "patterned after a television system" [22, p. 721]. Objective methods are used to test these characteristics and the "numerical values obtained by calculation...require correlation with the subjective impressions: graininess, tone scale, and sharpness" [21, p. 9]. While his work is pioneering and a great foundation for modeling EO systems, it has been proven to be "complex and difficult to adapt to changing conditions" [21, p. 10]. Figure 2.8 illustrates a soldier looking through an image device to assist in his/her ability to assess whether an object is a threat or not. Not only is the human eye itself complicated, but adding to this complexity is trying to model photons hitting an object like a tank, bouncing into a device (UAV sensor), which in turn takes that image and makes a duplicate image. However, the M&S community is asked to come up with ways to best represent this real phenomena in simulations.

2.4.1 Johnson Criteria

Johnson Criteria assists analysts in understanding the critical dimensions necessary for an observer to "see" a target. The United States Army Night Vision and Electronic Sensors Directorate (NVESD) state: "John Johnson, a Night Vision scientist, worked to develop methods of predicting target detection, orientation, recognition, and identification. Johnson worked with volunteer observers to test each individual's ability to identify targets through image intensifier equipment under various conditions" [23]. His models allow the ability to create predictive models of imagery systems. Johnson believed that alternating lines of a fixed width and spacing enable one to characterize the level of detail that can be discerned from an object. The thinner the line and closer the spacing, the more detail a person can perceive. He developed thresholds of line pairs to predict a person's ability to perform detection, orientation, recognition, and identification [21, p. 7]. The Johnson Chart can be found in Appendix A, which shows the minimum number of line pairs for the probability of 50% of observers to obtain detection, orientation, recognition and resolution when looking at the target. Figure 2.9 illustrates the number of pixels needed for detection, recognition and identification of a human being. It makes sense that the level of detection, recognition, and identification is proportional to the number of pixels that cover the respective area of acquisition. Vollmerhausen and Jacobs [21, p. 7] also state:

> The Johnson metric uses limiting bar-chart resolution as an indicator of sensor goodness for target acquisition purposes. Predictive accuracy of this metric is best when comparing "like" sensors and conditions. The metric is not compatible with many features found in modern sensors. For example, it is not compatible with sampled imagers. Further, the Johnson metric fails to predict the impact of frequency boost on range performance.

The Johnson Criteria allows for the modeling of target acquisition predictability for different sensors currently in use by the U.S. Army. The metric does have its limitations that may inhibit accurate prediction of what humans will see when using imagery systems. The ACQUIRE model performs target acquisitions in COMBATXXI and requires the Johnson Criteria critical dimension for target detection. Since the Johnson Criteria are in need of an update to reflect the effects of STA when operating UASs, it may be beneficial to understand more about ACQUIRE and possibly construct a model without using the Johnson Criteria.

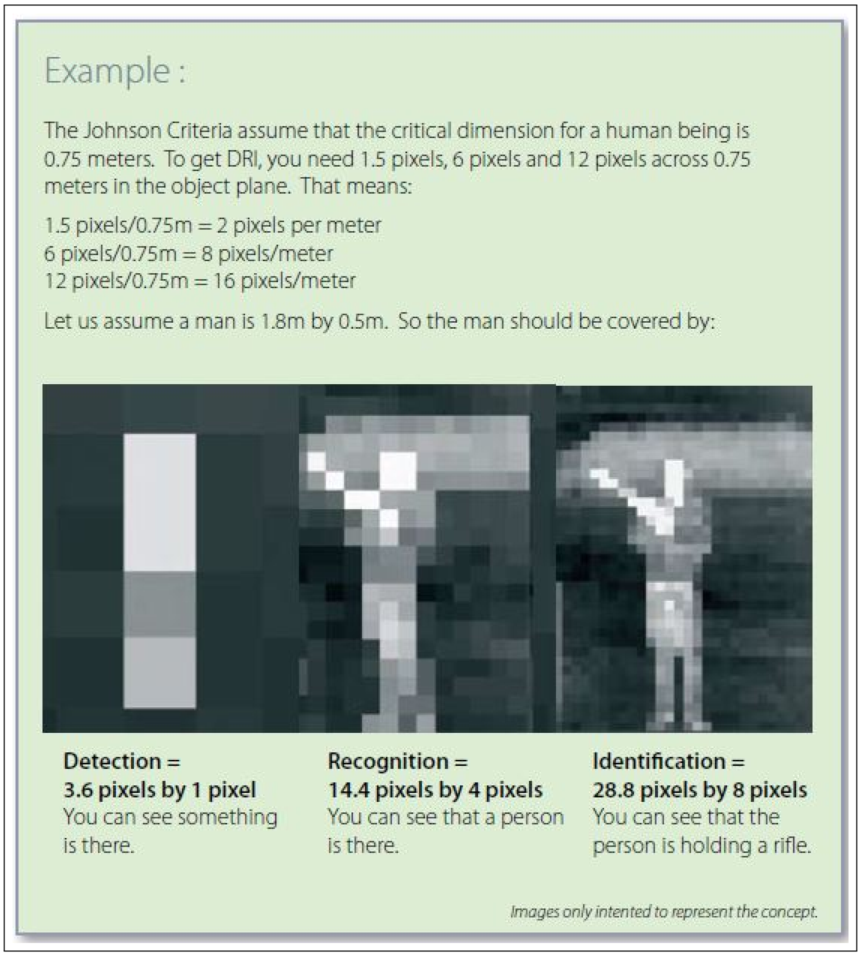

Figure 2.9: Illustration of number of pixels required for detection, recognition, and identification of a human being. From [24, p. 3]

2.4.2 ACQUIRE

The most prevalent target acquisition model the U.S. Army operates in combat simulations is the ACQUIRE model. The ACQUIRE model was formulated in 1990 and is a continuation of the Johnson Criteria [25, pp. 1-2] According to Darken [26, p. 263] “ACQUIRE computes the detection probability as a function of the brightness...of the target, the brightness of the background of the target, and the subjective size of the target, in terms of its 'number of resolvable cycles.'" The ACQUIRE model uses physical sensor characteristic data and human perceptual data along with environmental factors to compute a probability of detection for a specific observer-target configuration. In combat models such as COMBATXXI, this data coupled with a stochastic process allows for determining an actual outcome of a simulated observation. For example, ACQUIRE computes the probability that one will see a target given a target's range and the entity's sensor as .78. COMBATXXI then randomly draws a (uniform) number from 0–1 (where all values between 0 and 1 have an equal chance of being selected). If the number drawn is equal or less than .78 the simulation says that target is "seen" otherwise, the target is not seen even though there is clear line of sight between the sensor and the target.

LTC Baez [27] identifies some major drawbacks to the ACQUIRE model:

- "Current [model has] not been calibrated or developed for close-in targets within 200 meters" [27, p. 5].
- "Simulated soldiers can scan large fields of regard (FOR) quicker and make significantly quicker detections and identifications than a real soldier" [27, p. 5].
- Unvalidated workarounds have been made for limitations such as moving targets, multiple targets, clutter effects, color effects.
- Lack of updated visual perception experiments for targets not in database (i.e., components of IEDs).

These four items listed above hinder the ability to accurately predict the detection of targets by a UAS modeled in this study. The UAS that will be modeled in COMBATXXI for this study is the Raven RQ-11B. The Raven generally operates between altitudes of 30–152 meters [7]. When operating the Raven soldiers scan the hand-held screen and probably do not always detect (within a reasonable amount of time) objects such as a person holding a cell phone, misplaced box along the road, wires coming out of the ground. To compound the problem, there are no known human experiments that have been run to accurately model the detection of items using a sensor such as this UAS.

2.5 COMBATXXI

The Combined Arms Analysis Tool for the 21st Century (COMBATXXI) is a detailed event-driven simulation developed by U.S. Army Training and Doctrine Command White Sands Missile Range (TRAC-WSMR) and Marine Corps Combat Development Command (MCCDC). The COMBATXXI User's Guide describes this simulation as "A Joint, high-resolution, closed-form, stochastic, discrete event, entity level structure analytical combat simulation" [1, p. 14]. Some of the major model functions include "Ground Combat (Light and Heavy Forces), Future forces, Fixed-wing and rotary wing" [1, p. 14]. One of the goals of COMBATXXI is to provide "a simulation that can represent information flow in a way that allows the analysis of its impact on operational effectiveness" [1, p. 14].

Behaviors of entities in COMBATXXI can be specified by model users using [1, p. 317]:

- Orders
- BSL (Behavior Specification Language, the native scripting language in COMBATXXI).
- Python scripts

Orders are the easiest for the programmer to use and are grouped into compound orders to perform more complex actions. BSL provides more complex behavior structures and a way to connect to complex internally defined behaviors. However, BSL restricts access to certain objects and data outside the model, which limits its use of making complex, dynamic behaviors. Python allows for the most flexible method of developing dynamic, "realistic" behaviors in COMBATXXI. To the non-programmer this may be seem a little intimidating. Figure 2.10 relates the types of behavior being programmed to the difficulty of implementation. The next section provides information on how this study will utilize dynamic planning methods and tools to assist in creating these dynamic behaviors.

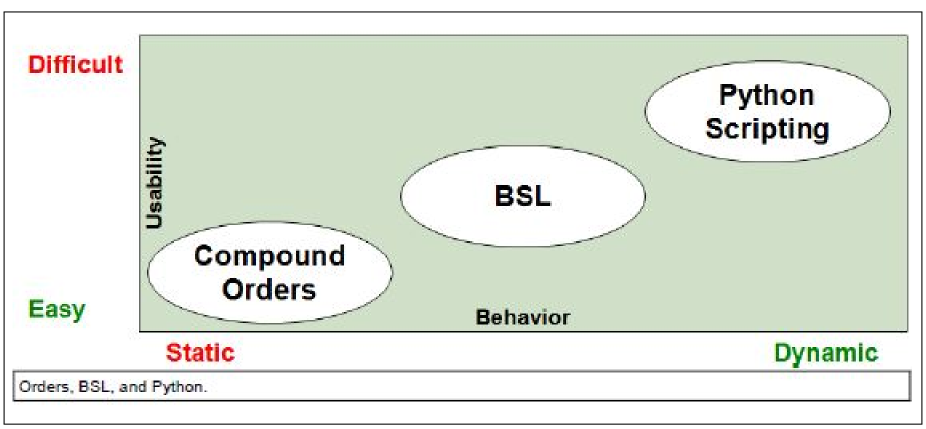

Figure 2.10: Relationship of behavior (static, dynamic) vs usability (easy, difficult) when considering different programming methods in COMBATXXI. From [1, p. 317]

HTN Tasks and Shapes	
Tasks	Shape
Primitive Task	
Compound Tasks	
Goal Tasks	
Constraint Node	

Table 2.1: HTN Task and Nodes. From [30]

2.6 HIERARCHICAL TASK NETWORKS

The father of modern-day hierarchical task networks is Earl Sacerdoti, who describes HTNs as "procedural nets" [28]. HTNs are a means of planning actions in a program via the traversal of a series of networks in order to reach a goal. Sohrabi, Baier, McIlraith [29] state: "the planner is provided with a set of tasks to be performed, possibly together with constraints on those tasks. A plan is then formulated by repeatedly decomposing tasks into smaller and smaller subtasks until primitive, executable tasks are reached." Traditional HTNs consider what is to be the desired state of world, finds the plan (based solely on the current state of the world) which then is executed to reach the goal state (the desired state of the world).

2.6.1 Basic Terminology of HTNs

In order to fully appreciate the potential of HTNs, some basic terminology must be defined. *Primitive tasks* are tasks that cannot be broken down any further. A *compound task* consists of a collection of tasks that may be primitive or complex. *Goals* or *goal tasks* represent the final world state the HTN is trying to achieve. *Constraints* are the conditions that must be true in order to execute a specific branch of the HTN. See Table 2.1 for the graphics that correspond to an HTN task and/or nodes.

HTN trees are read from left to right and then top to bottom. Traversal of the HTN is complete once the goal node is reached. If the goal node is not reached, then the HTN might not finish and thus the goal state is never reached. Proper HTN fabrication allows for goal nodes to always be reached [3]. In the next section, an example of a static HTN in operation provides an opportunity to understand traditional automated planning.

2.6.2 Example of Static HTN Planning

A HTN creates a plan to implement the behavior of someone going to the store (see Figure 2.11). The plan is based on the current state of the world and is valid as long as the state of the world does not change; remains static. The traversal of this tree illustrates the methodology of HTNs. First, the "Go to Store" compound task begins execution. Next it moves to the first and only constraint "Find Keys." If "Find Keys" has value of "True" then the tree traverses to the first primitive (non-compound) task "Get in Car." After this task the program automatically executes "Drive to Store." Once "Arrive at Store" executes, the goal has been reached and the HTN terminates. Going back to the "Find Keys" constraint node, if one does not have keys (value of "Find Keys" constraint has value of "False") then the "Walk to Store" primitive task is reached and then "Arrive at Store" performs and the HTN terminates. Using HTNs this way is great if and only if the conditions (known state of the world) have not changed since the plan was formulated and execution was started. But cars do break down and/or run out of gas. Road construction causes detours resulting in alternating course (changing plan) in order to ensure that the goal "Go to Store" is met. To address these types of problems, modifications need to be made to the HTN methodology. In the next subsection the use of dynamic planning is discussed.

2.6.3 Example of Dynamic HTN Planning

If the state of the world changes after a developed plan starts execution the entity may never reach its goal. To account for a possible change, the programmer may add an interrupt goal to force the HTN planner to reassess and replan actions to ensure a feasible plan is made. Interrupt nodes for this demonstration are labeled red. Balogh et al. state that interrupt nodes serve two purposes: "represent tasks that takes some time to complete and are likely places where the plan become invalid and allow for the 'lazy generation' of the plan only part of the plan needs to be generated" [3, p. 4]. In addition to the interrupt node, the idea of a replan event needs to be introduced. A "replan event" will cause the tree to "suspend execution of the current plan and reevaluate the HTN being used to determine how the plan should be changed based on the current state of the world" [30, slide 13].

Looking at Figure 2.12, it displays the "Go to Store" task with the addition of interrupt goals. For this example the idea is that one is at home (at the start of execution) and along the way to the store, a problem will arise causing the planner to account for this unexpected event, which would normally cause the plan to be void and the goal never met. The replan event indicates

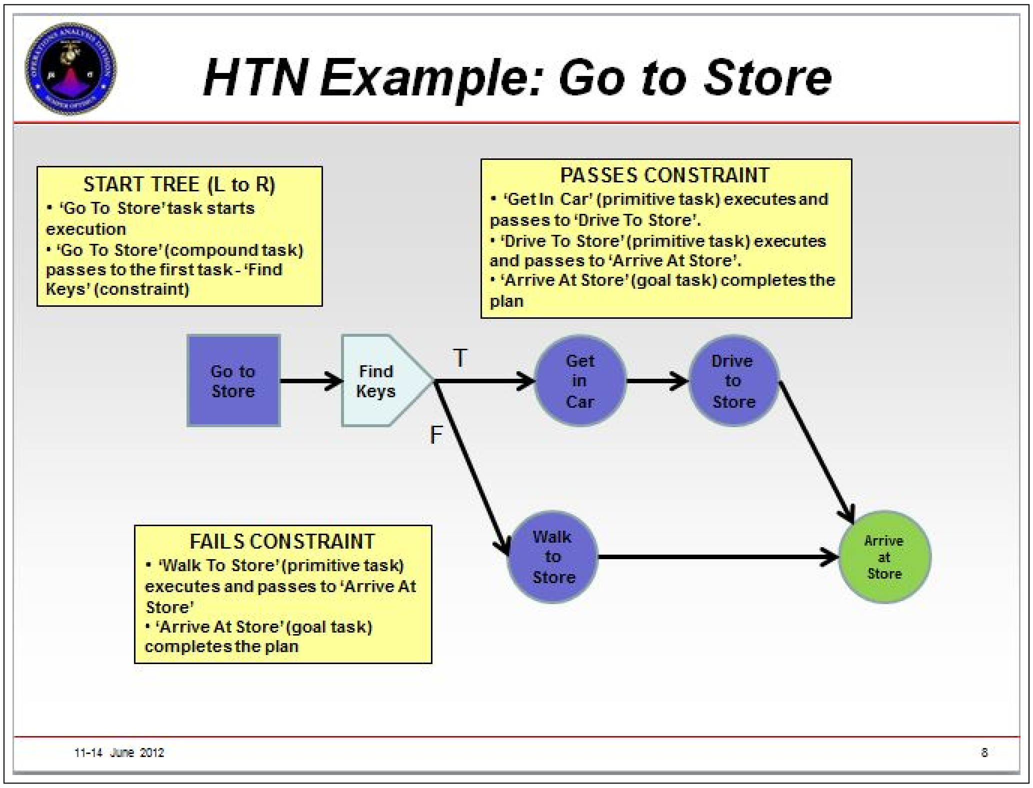

Figure 2.11: Planning to go to store without accounting for changes in the environment. From [30, slide 8]

that a state is reached where a change in the state of the world could occur and the whole plan will re-evaluate. When an interrupt node is hit, the execution of the tree halts, pending a replan event causing the tree being reevaluated. The replan event for this tree is "Stopped Moving." First the "Go to Store" compound task executes and reaches the "At Store?," which has a value of "False." Next the "At Home?" (interrupt node) has value "True" and tree planner moves to constraint node "Find Keys." Since that is true "Get in Car" task is planned, which then triggers the interrupt node "Drive to Store," causing the planner to stop executing. Since the replan event "Stop Moving" was triggered after "Drive to Store" event executed, the tree re-executes at the root node "Go to Store." The tree looks at constraint node "At Store?," whose value is "False," then moves to "At Home?" constraint node whose value is also "False" then plans an "Walk to Store" node. The "Walk to Store" node triggers the "Stop Moving" event causing the tree to be executed again. The task "Go to Store" executes and moves to constraint node "At Store?" being "True" causing the "Arrive at Store" goal node to execute allowing tree to be completed and finished. The interrupt node allows the tree to stop executing to allow for replan triggers to cause the tree to form a new plan in order to reach a goal.

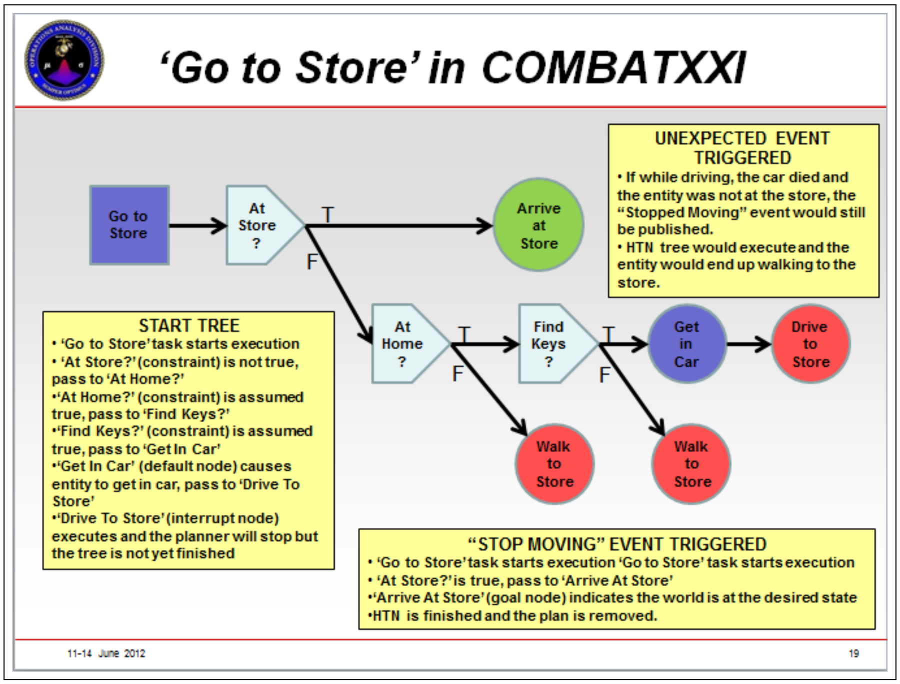

Figure 2.12: Going to store with an interrupt node. From [30, slide 19]

2.7 KNOWLEDGE REPRESENTATION

Information can be pieces of fact and fiction that describe the world around us. Knowledge is the collection of information that allows for intelligible action to take place. It is imperative that one knows how to apply knowledge and relate it to ideas that can accurately reflect the message one is trying to convey. Brachman and Levesque say knowledge representation is "the field of study concerned with using formal symbols to represent a collection of propositions believed by some putative agent" [31, p. 4]. Davis, Shrobe, and Szolovits see knowledge representation (KR) as "a surrogate, a substitute for the thing itself, that is used to enable an entity to determine consequences by thinking rather than acting that is, by reasoning about the world than [*sic*] taking action on it" [32, p. 17]. They explain one of the most interesting aspects of reasoning is that one must decide what the surrogate represents and what level of fidelity is the surrogate given when describing the entity [32, p. 18]. Their example consists of an entity planning on assembling a bicycle. There are different parts of the bicycle the person would have to think about (internally) that exist (externally) [32, p. 18]. Ways of representing knowledge via computers have been around for over fifty years. Some means of KR have included "neural networks, theorem proving and expert systems" [33]. In order to communicate an idea effectively, one needs to be able to speak in a manner that both parties can understand (a common language). There are languages (old and new) that have been adopted and created to help present the understanding of the world. A unique problem with Artificial Intelligence (AI) is that a program needs to have precise understanding of what the digit "2" is versus the string "too." We, as humans, must know how and when to use "2" and "too" when talking to one another. For example, concepts such as aircraft, flying, and travel can be closely related or very far apart. It all depends on the idea that you are trying to represent. If one is trying to plan a trip to Disneyland the first thought is the modes of "travel." Travel could happen by driving, sailing, swimming (if you are a good swimmer) or flying. By flying, concepts such as "soaring," "being lighter than air," "flapping of wings" may come to mind. However, since the context is in the realm of transporting a person from their home town to Disneyland, it is most likely that "flapping of wings" is not the true method of flying in this scenario. A depiction of being enclosed in a fuselage resembling something like a Boeing 737 or Airbus 350 and soaring through the clouds may be more accurate. The aircraft engineer might visualize movement of air molecules over the wings creating lift opposing the downward force of gravity. There are infinite ways to represent knowledge, unlike many problems that have a constrained solution space. Below are some areas that have been used to formally represent knowledge.

2.7.1 Propositional Calculus

Propositional calculus (aka propositional logic) is said to have been first created by the philosopher Aristotle [34]. In general, it is made up of symbols, truth symbols, and connectives. Propostional logic "is the branch of logic that studies ways of joining and/or modifying entire propositions, statements or sentences to form more complicated propositions, statements or sentences" [34]. These propositional sentences declare if the proposition is true or false [34]. Well-formed formulas (WFFs) are legal sentences in PL. Table 2.2 shows the symbols and well-formed fumulas. Propositional logic is concerned with the assigning of truth-values to a proposition (true or false). A PL sentence cannot be both true and false. The "Internet Encyclopedia of Philosophy" provides a very thorough review of Propositional logic [34].

2.7.2 Predicate Calculus

Predicate calculus allows one to tie relationships together and make clearer connections to objects, persons, ideas,...etc. Sentences are formed in predicate calculus just like they are in propositional calculus using the same connectives as found in Table 2.2. In propositional logic the atoms "P" and "Q" represent a proposition that only has meaning to itself and itself alone.

Symbols of Propositional Calculus		
Propositional Symbols	P Q	"The sun is shining." "Today is Friday."
Truth Symbols	true false	"True" "False"
Connectives	$\neg$ $\wedge$ $\vee$ $\rightarrow$ $\equiv$	negation, "not" conjunction, "and" disjunction, "or" implication, "if...then..." equivalence, "equivalent to"
Well-Formed Formulas (WFFs)	P, Q, R $\neg$ P P $\rightarrow$ Q P $\wedge$ Q $\equiv$ R	Propositions "not P" "P implies Q" "P and Q is equivalent to R"

Table 2.2: Propositional Calculus Symbols.

However, with predicate calculus, values can be assigned to the propositions individually, which allows the ability to make new inferences by explicitly referring to the values assigned to the symbols. If "X" is a variable representing hours of the day and sentence "weather(X, sunny)" is true then literally one is expressing "the weather is sunny all the hours of the day." Two new symbols are introduced in predicate calculus: the universal quantifier, $\forall$, and existential quantifier, $\exists$. The universal quantifier states that a sentence is true for all values of the variable. The existential quantifier states that there exists at least one value of the variable that make a sentence true. Luger states, "A major challenge for AI programmers is to find a scheme for using these predicates that optimizes the expressiveness and efficiency of the resulting representation" [35, p. 60]. Table 2.3 shows the power of representing "knowledge" using predicate calculus. Predicate calculus allows for a formalized fabrication of information enabling the structuring of knowledge bases, which will be shown later in the thesis.

2.7.3 Semantic Networks

A semantic network can be used to represent knowledge by using meaningful relationships with nodes and arcs. According to associationists, "when humans perceive an object, that perception is first mapped into a concept. This concept is part of our entire knowledge of the world and is connected through appropriate relationships to other concepts" [35, p. 229]. AI pioneers Collins and Quillian fabricate a semantic network based on human information and storage times. It is said that by using inheritance, humans are able to decrease the size of needed knowledge

Predicate Calculus and the English Language	
If it does not rain on Monday, Tom will go to the mountains.	¬weather(rain, monday)→go(tom, mountains)
Emma is a Doberman pinscher and a good dog.	gooddog(emma)∧isa(emma, doberman)
All basketball players are tall.	∀ X (basketball_player(X) →tall(X))
Some people like anchovies.	∃X (person(X, anchovies))
If wishes were horses, beggars would ride.	equal(wishes, horses)→ride(beggars)
Nobody likes taxes.	¬∃X likes(X, taxes)

Table 2.3: English sentences being used in predicate calculus. After [35, p. 60]

base by this form of abstraction. Figure 2.13 shows a Collins and Quillan hierarchy of two distinct type of birds, characteristics of birds, and how they both relate to animals representing the idea of inheritance. A node could represent concepts such as objects, ideas, or situations. The arc represents the relationship between two objects, ideas or situations. Figure 2.13 shows a representation of how human information could be stored in memory of a computer via a semantic network. This is why it is important to understand how a semantic network can be built to better understand beings, ideas, and the relationships they share.

2.7.4 Ontologies

An ontology is a way of creating our knowledge base (KB) of the world and having the agent of interest act upon the knowledge it knows of the world and make new inferences. We can use predicate calculus to help create the "facts"' of our known world. *Merriam-Webster* defines ontology as "a branch of metaphysics concerned with the nature and relations of being" [37]. Brachman and Levesque state "[ontology represents] the kinds of *objects* that will be important to the agent and the *properties* those objects will be thought to have, and the *relationships* among them" [31, p. 32]. There are software programs that can assist in developing simple to rather complex ontologies that would be otherwise impossible for a human to store in his or her brain. The Web Ontology Language (OWL) is a high-level knowledge-based language used to construct knowledge base for many domains. It allows for machines (computers) to automatically process information based on the predicate's expressiveness. Childers discusses the benefits of using ontologies "[because ontologies]...allows programs to understand, process, and infer new information from coherent data" [38, p. V]. An ontology should assist in the creation of a knowledge base enabling a better understanding of the importance of acquiring accurate and useful information such from sensors in a UAS.

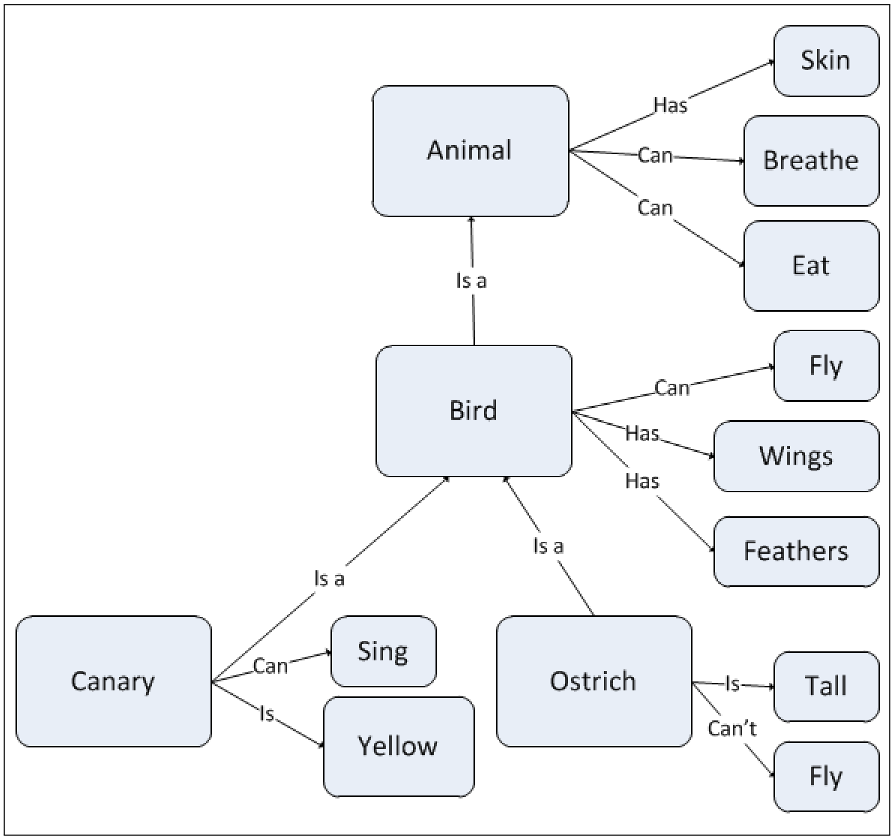

Figure 2.13: Semantic Network from Collins and Quillian modeling how information would be stored in a computer. This is a modified remake of their illustration. From [36, p. 241]

2.8 SUMMARY

The DoD has use for many types of UASs, of which one is reconnaissance. There are methods to model the sensors and sensing systems on these aircraft, however, they have drawbacks. In modeling, there are many ways to represent knowledge and decision making.

CHAPTER 3: METHODOLOGY

3.1 INTRODUCTION

The methodology begins with developing a scenario that implements the Raven UAS in an typical Southwest Asian (SWA) environment. HTNs form the basis for planning and creating behaviors of the entities in the scenario. The creation of design of experiments and an ontology assist in measuring the effects of UAV operator skill and decision making.

3.2 SCENARIO

In order to understand the capabilities, limitations, pros and cons of an UAS, it is imperative that we consider the context of its usage. For this thesis, the scenario takes place in Southwest Asia. The 5-Ws (who, what, when, where and why) give the reader the understanding of what kind of mission will take place. Below is a brief description of the scenario [39] to be modeled in COMBATXXI. It represents a scenario that a small unit living on a combat outpost (COP) may have to perform while conducting operations with a host nation's army in SWA (see Appendix B).

- Who: Squad of U.S. Forces from COP Ramrod consisting of 7 vehicles and 1 RQ-11B RAVEN UAS.
- What: Conduct Resupply Run to Outpost X-Ray
- When: 0800 Hours Local Time
- Where: On an unimproved route consisting mainly of dirt and some gravel.
- Why: To resupply a joint American/Host Nation Outpost consisting of approximately 30 personnel.

Along the route there are areas known to have a higher than normal likelihood of containing an IED. There are four named areas of interest (NAIs) that have an equal chance of having an IED. For this study the real IED can be found in NAI # 2. The Raven UAS provides the added ability for the convoy commander to search the area at a safer distance than without a Raven. In the study it is assumed that a convoy will not see signs of a possible IED until they are within close proximity of one (approximately 50 meters). The mission can be seen in Figure 3.1 with NAIs and the convoy's start and end points labeled.

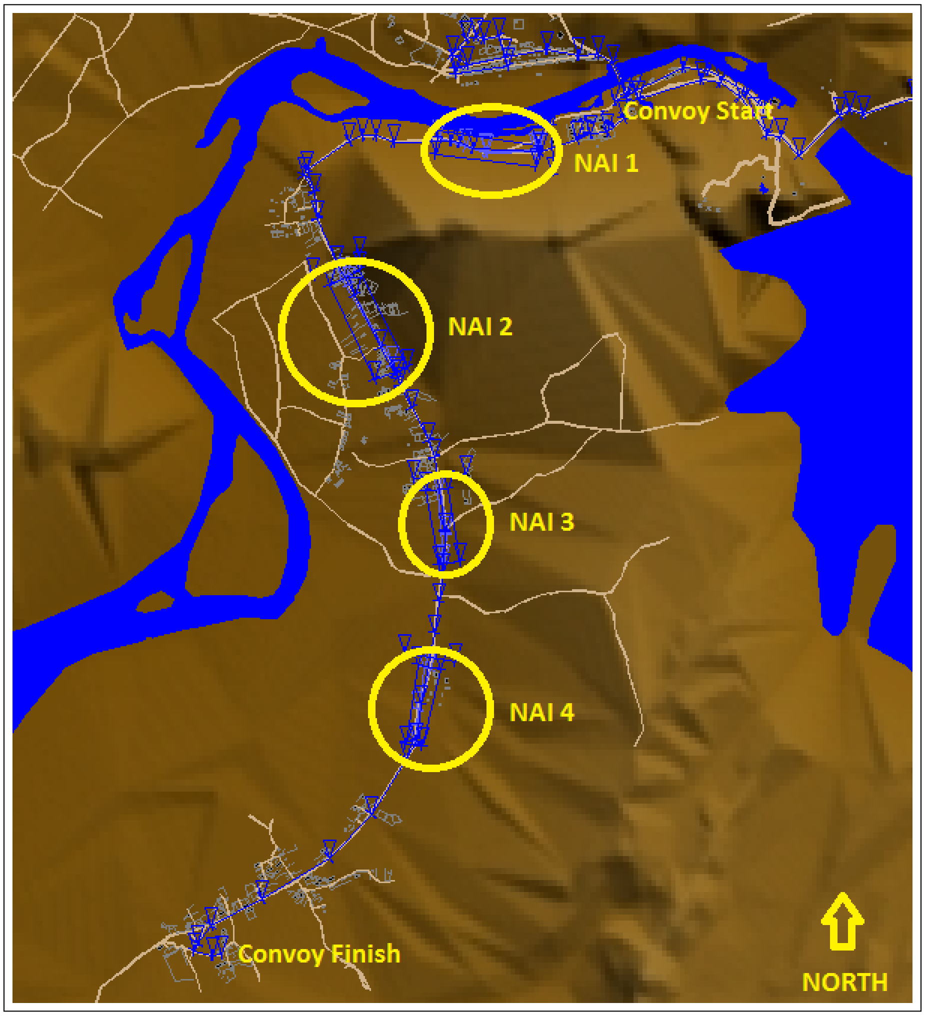

Figure 3.1: Mission Overlay in COMBATXXI.

When the convoy passes each phase line (PL) at the start of each NAI, they will stop and deploy the Raven. It will search each NAI by flying along a pre-determined path in search of an IED, see Figure 3.2. An object representing the IED will only detonate with an assigned standard uniform probability of .8 given that a convoy entity enters IED's sensor range of 10 meters. If an object is outside of the 10 meter range, the IED will never activate. The Raven UAS will defeat the IED if 1) the UAV detects the object that "could" be an IED (based on new UAV sensor model) and 2) the operator believes that the object detected *is* an IED and takes appropriate action by calling the unexploded ordnance clearing team (EOD). Calling EOD will always result in a 60-minute time penalty and IED being destroyed. If the convoy is hit by an IED, then mission is over. Success is the ability for convoy to safely arrive without taking any casualties.

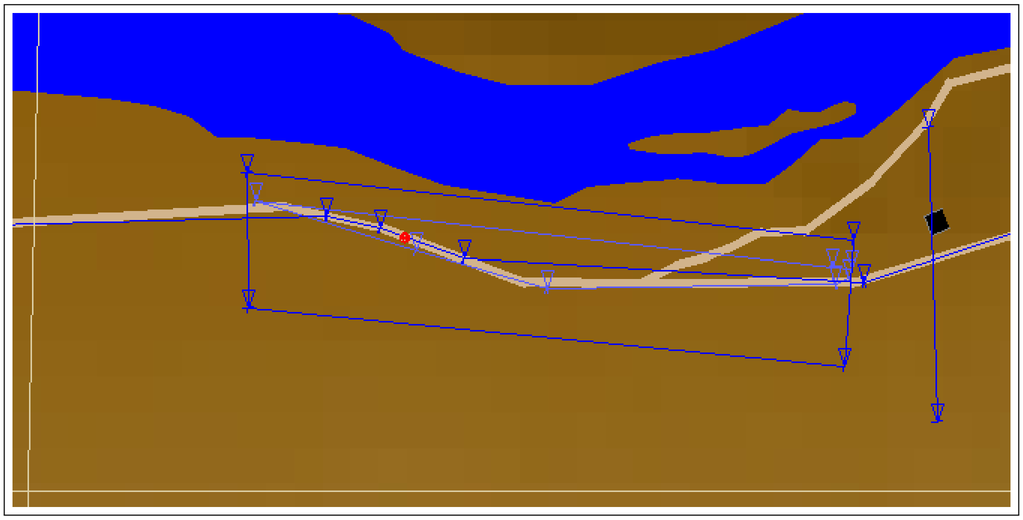

Figure 3.2: Screen shot of NAI 1.

Next, the research will look in to developing behaviors of our entities in the COMBATXXI scenario.

3.3 FORMULATION OF BEHAVIORS

The study uses HTNs to define behaviors for the entities modeled in the research. A tool called "Behavior Studio," developed by the MOVES Institute, provided for formation of behaviors. HTNs promise to make coding behaviors less time intensive, more efficient, etc., and in doing so allow for quicker formation of behaviors to be analyzed. An HTN tree is one separate entity that has a root node and at least one goal node. Each tree is organized with an initialization branch and execution branches. The execution branches consist of **`events`** and **`modelEvents`**. The events generally consist of events that are organic to the planner, whereas modelEvents plans for the execution of events directed by COMBATXXI exclusively. The list of trees for this study are as follows, with a broad description of the behavior to be modeled.

- **`ConvoyMoveInFormation`**: Behavior of moving convoy along the route.
- **`OperateIED`**: Behavior enabling detonation of IED when appropriate.
- **`OperateUAS`**: Behavior for operation of the UAV and convoy.
- **`UAVSensor`**: Behavior of the Raven UAV sensors.

3.3.1 Moving the Convoy HTN

The behavior tree labeled **`ConvoyMoveInFormation`** is responsible for ensuring the convoy reaches its final destination (Outpost X-Ray), see Figure 3.3. When the tree executes it

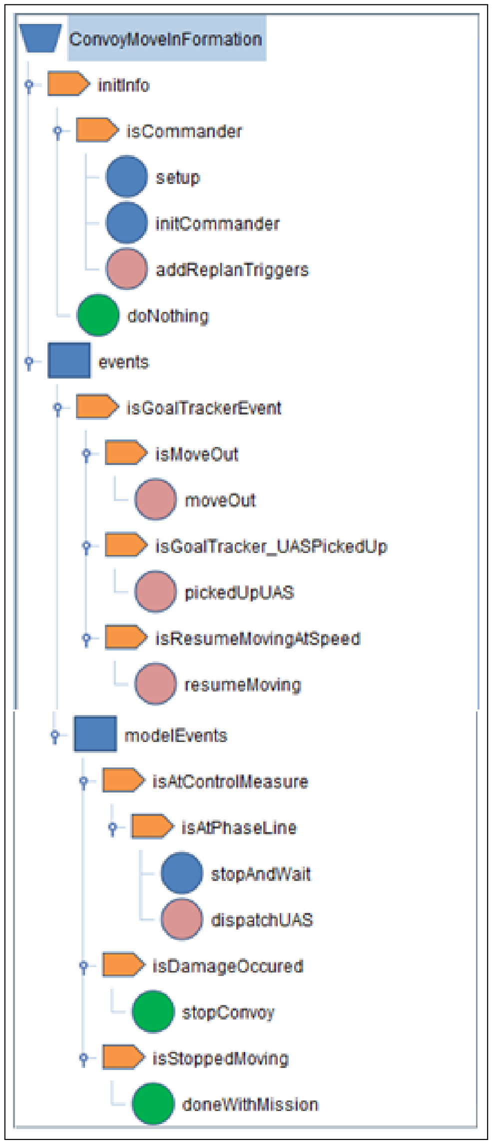

Figure 3.3: **ConvoyMoveInFormation**Tree. This tree will form a plan to get the convoy from the COP Ramrod to OP X-Ray.

first "initializes" using the **initInfo** and runs to the first condition **isCommander**, which tells that there are the NAI phaselines to pay attention to and schedules a "Move Out" event. Then several replan triggers are added for the tree listen to via **addReplanTriggers** node. The **doNothing** node tells the tree to end planning if someone other than the commander is assigned this tree. Once the **initInfo** branch completes, the execution is interrupted and is waiting for a replan event.

When a replan event occurs different nodes may execute. The **moveOut** node orders the convoy to move out. The **pickedUpUAS** interrupt node informs the convoy commander if the UAS (Raven) is physically back with convoy. If the UAS has not seen any artifacts that could be dangerous (such as potential IEDs) it schedules an "ResumeMovingAtSpeed" event. If the condition **isResumeMovingAtSpeed** is true the plan passes to the **resumeMoving** interrupt node that has the convoy resume normal convoy speed.

Next, moving on to **modelEvents** node, if conditions are true in nodes **isAtControlMeasure** and **isAtPhaseLine** then the convoy will come to a "halt" (**stopAndWait**). Raven dispatches in search of IEDs, respectively. The **dispatchUAS** interrupt node will assign the **OperateUAS** tree to the the UAV entity to execute and search the current NAI.

If the **isDamageOccured** node is "true," then the convoy has been hit by an IED, meaning the mission is over and the plan terminates appropriately. And finally, if the convoy has reached its final destination, the **doneWithMission** node is executed and the tree terminates. The complete **ConvoyMoveInFormation** tree can be seen in Appendix C.

3.3.2 Operating the IED HTN

The HTN tree labled **OperateIED** forms the behavior of the improvised explosive device. Figure 3.4 shows the tree in its entirety. First the tree jumps into the **initInfo** node and checks to see if the object is a real IED (**isRealIED**) by checking to see if probability of detonation is greater than 0. If object is not an IED the tree moves to goal node **setIsNotIEDInfo** and finishes. However, if **isRealIED** condition is "true" the tree moves to the **startObservation** node, activating the IED if convoy entity is within the activation (sensor) range of IED. Next the **addReplanTriggers** executes and ensures **OperateIED** tree is replanned if entities enter IED's area of interest (AOI), the simulation publishes a "GoalTracker_Detonate Now" event. Now moving along into the **events** branch, the IED detonates as long as "doGoalTracker_DetonateNow" is true. The IED will detonate with a probability (0.8 for the study) passed in as a parameter and is ALWAYS catastrophic (no survivors). The **modelEvents** node schedules a replan trigger called "GoalTracker_DetonateNow" when the second vehicle enters the IED's killzone. The complete **OperateIED** node can be seen in Appendix D.

3.3.3 Operating the UAS HTN

The HTN tree **OperateUAS** forms the behavior of operating the UAS, see Figure 3.5. Like earlier HTNs, the tree first initializes and adds replan triggers that will cause the tree to replan when the UAS is finished searching or scanning the NAI. Next, the tree plans routes for

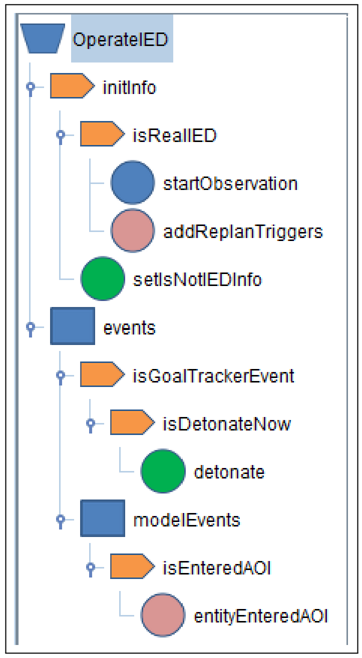

Figure 3.4: **OperateIED**Tree. Picture of **OperateIED** HTN.

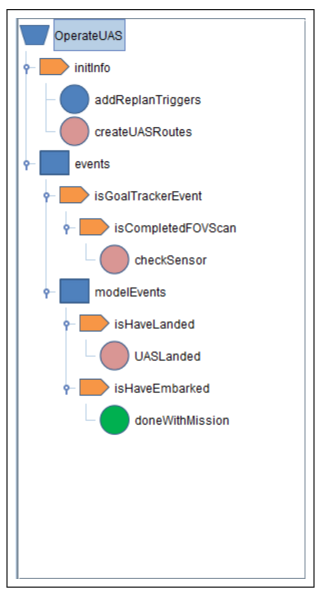

Figure 3.5: **OperateUAS**Tree. Picture of **OperateUAS** HTN.

the UAV to operate, see interrupt node **createUASRoutes**. Moving to **events** node, the branch is responsible for checking the sensor for objects found once it finishes scanning. The **isCompletedFOVScan** checks if UAV scanning its field of regard (FOR). If **isCompletedFOVScan** has value of "true" then the interrupt node **checkSensor** executes. In this node, if the UAV

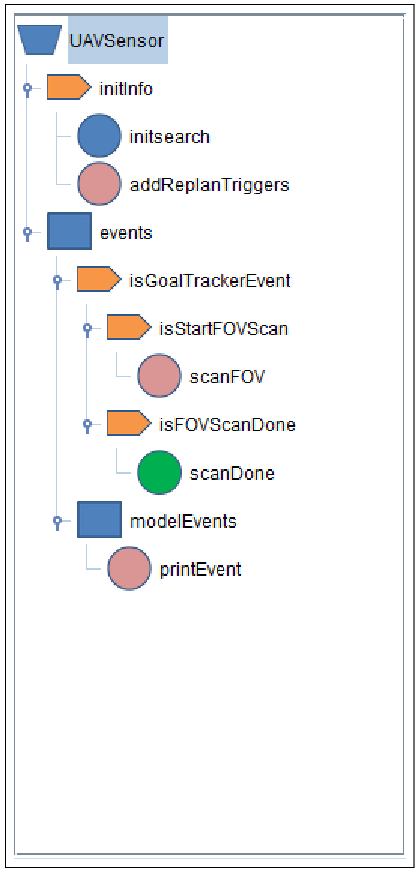

Figure 3.6: **UAVSensor** Tree.

entity has not returned to the convoy or is greater than 100 meters out from vehicle it disembarked, the tree will push the UAV sensor tree to **UAVSensor** tree where the UAV entity continues its search of FOR with necessary parameters.

When the tree traverses down to the **modelEvents** once the constraint node **isHaveLanded** is true, the **UASLanded** node executes–meaning that the Raven UAV has landed. Lastly, once the UAV embarks the convoy (it just completed its mission), then the goal node **doneWithMission** executes and the human operator determines if this is either an IED or not an IED. A more detailed depiction of the tree can be given in Appendix E.

3.3.4 UAV Sensing HTN

The **UAVSensor** HTN is responsible for modeling the sensor of our Raven UAV (see Figure 3.6). The goal here is to model a user observing through a single field of view. First the tree initializes and the **initsearch** node computes a scan time for the FOR (field of regard), which is simply what the person is focusing on the screen. The field of view (FOV) is the entire screen that the observer cannot see due to the eyes abilities to only focus on a few degrees at a time. Moving down into the `events` branch if the `isGoalTrackerEvent` node and `isStartFOVScan` node are true the interrupt node `scanFOV` executes. In this node, the UAV searches for objects on the ground using an EO sensor. It has the probability of detecting objects in its path based on the distance. Looking at the `isFOVScanDone` node is true, it passes to the goal node `scanDone`. This node finishes the tree and the "observations" will be looked at in the `OperateUAS` node (mentioned earlier) for determination if it is or is not an IED. A more detailed depiction of the tree in its entirety can be found in Appendix E.

3.4 UAV NOTIONAL SENSOR MODEL

To achieve the representation of a UAS modeling the effects of operator skill, we need a sensor. The sensor must be able to report to the `OperateUAS` HTN (representing human behavior) all possible IEDs. Currently the sensor in COMBATXXI is overly sensitive and cannot determine if an object found is an IED unless it is hard-scripted into that object as a real IED. Therefore the study uses a simple notational sensor developed by Imre Balogh (see Appendix F). The UAV sensor model can detect objects in COMBATXXI using some simple geometry and probability. Looking at Figure 3.7 one can see how the UAV detects an object. Probability of detection is based on the value of r. For this model the limit to detect an object is 220 meters. If the range of an object is 110 meters then the observer has a .50 chance of detecting object given that it is in the field of view of the sensor. Next, the study will look at the idea of creating a knowledge base (an ontology) that would assist the UAS operator in making decisions based on clues such as "disturbed earth," and "loose wires," instead of finding an object and flipping a coin to say that it is or is not an IED (as explained for COMBATXXI).

3.5 ONTOLOGY OF IED

Ontologies are a useful way of expressing knowledge of the world. They help to formalize an understanding of what objects will be represented and the relationships the objects share together [31, p. 32]. The agent in this study is the UAS and its ability to discern threat-type based on information received.

3.5.1 Developing an Ontology

This study seeks to develop a constraint that will allow our HTN to categorize the objects it observes. This categorization defines a state based on indicators in the ontology. This perceived state determines the selection of behaviors to follow in the HTN.

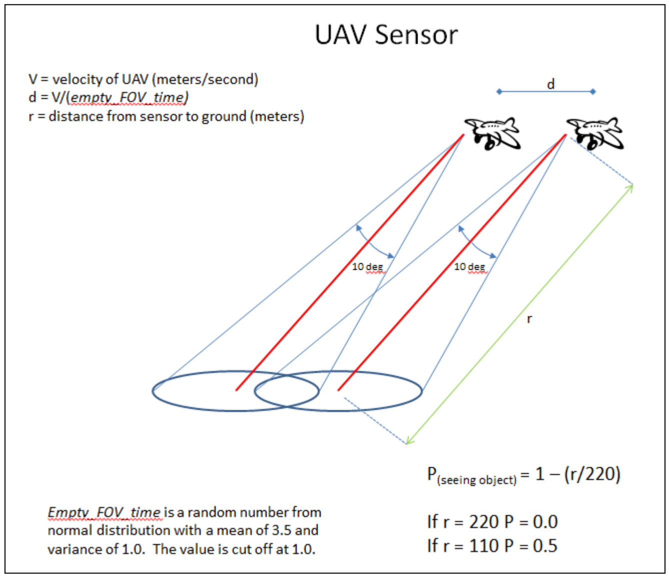

Figure 3.7: A depiction of the UAV Sensor Model used in this study. Courtesy from Imre Balogh [40].

This research uses the free, open-source program Protégé 4.3 to create a knowledge base (KB) of the world of IEDs [4]. According to the Protgé website, "Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats"[4]. There are two methods that ontologies can be formed: (1.) Protégé-Frames editor "*frame-based*, in accordance with the Open Knowledge Base Connectivity protocol (OKBC)" and (2.) Protégé-Owl editor, which "enables users to build ontologies for the *Semantic Web*, in particular the W3C's [(World Wide Web Consortium)] Web Ontology Language OWL" [4]. The W3C website states "The OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms" [41]. The study uses Protégé-OWL editor (version 4.3, build 304) to create the IED ontology. According to the Protégé-OWL website, "OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e., facts not literally present in the ontology, but entailed by the semantics" [4]. These logical consequences are derived by using the reasoner. A reasoner is "a piece of software able to infer logical consequences from a set of asserted facts or axioms" [42]. This study uses the Fact++ (fact plus plus) description logic (DL) reasoner.

3.5.2 Representing Information in an OWL Ontology

There are three main concepts when building an OWL ontology: classes, properties, and instances. The screen you see in Figure 3.8 shows the Protégé 4.3 user interface; the tabs used to create the ontology along with the reasoner are circled.

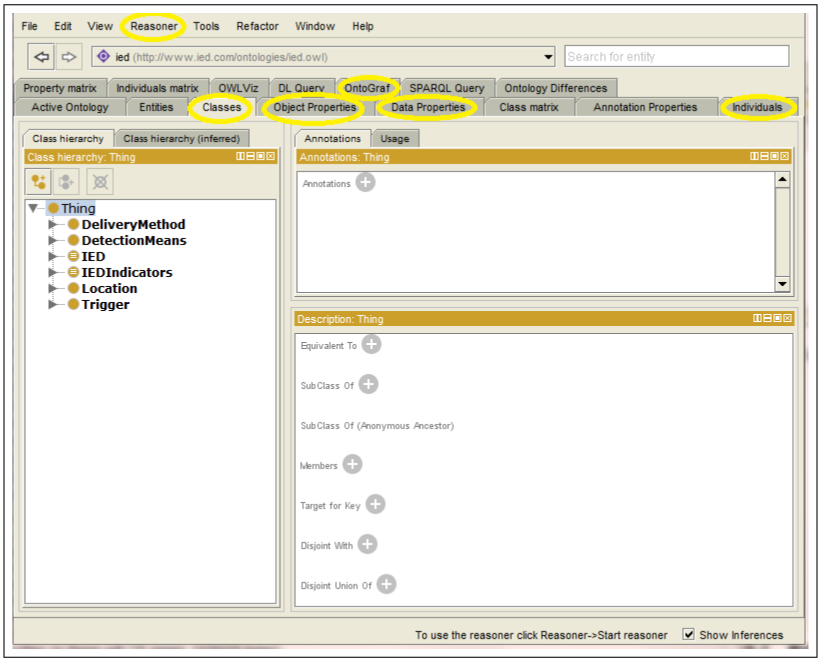

Figure 3.8: Screen picture of Protégé 4.3 program with the main tabs used circled. The Classes tab is shown.

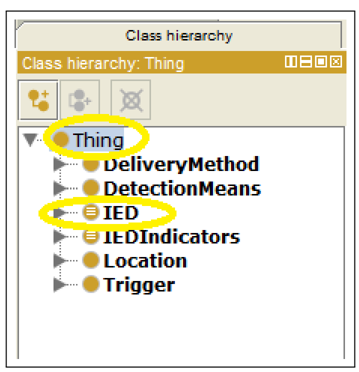

Figure 3.9: **`Thing`** class shown with its subclasses. **`IED`** class is a subclass of class **`Thing`**.

3.5.3 Defining Classes in IEDOntology

For this research, a preliminary ontology of IEDs was developed. The ontology is not fully defined, but provides a limited set of class and property definitions to demonstrate use of the ontology and automated reasoning to make inferences about detected IEDs based on information reported from human or automated observers in the battlespace.

In the Ontology (see Appendix G), the main classes can be seen in Figure 3.9.The **IED** class is shown (circled) with the other classes in the IED ontology. The main class is always the **Thing** class [43, p. 15]. The ultimate goal is to classify something as being some type of "IED," but in order to have a better understanding of the makeup of an IED, it is imperative that its sibling classes be defined first. The first sibling of class **IED** is the **DeliveryMethod**, which defines

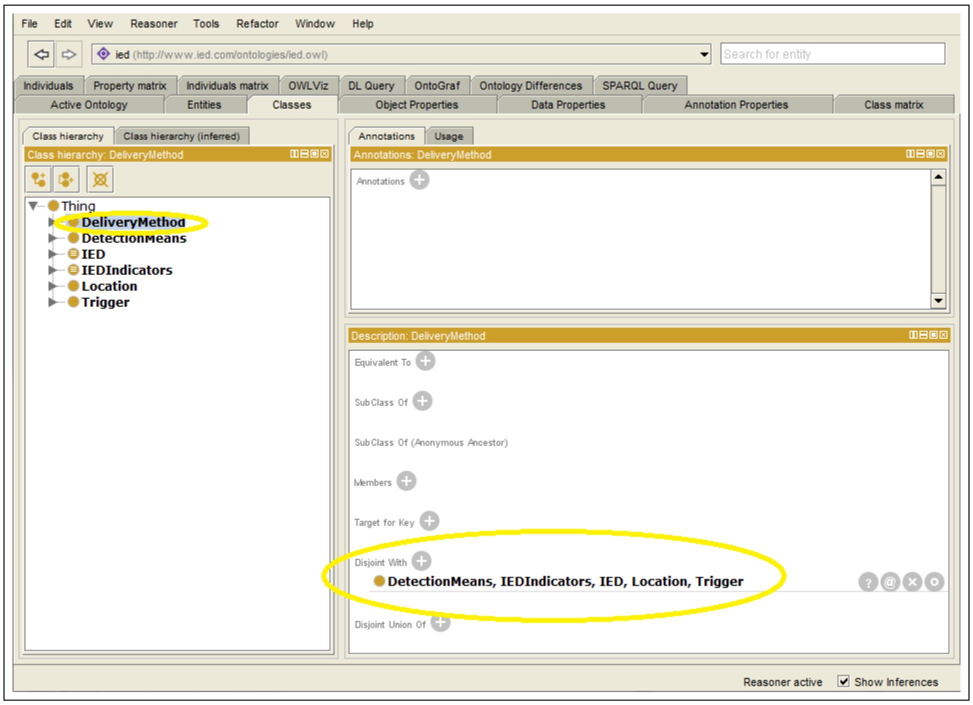

Figure 3.10: **DeliveryMethod** class shown with its disjoint sibling classes.

delivery method of an IED to its intended target (via human, package, or vehicle). The classes **Human**, **Package**, and **Vehicle** are subclasses of the parent class **DeliveryMethod** and are siblings to each other. This class is marked as disjoint from its sibling classes in the class hierarchy, as shown in Figure 3.10. We do not make them disjoint because the reasoner will fail. We make them disjoint because they truly are disjoint from each other. That information is used by the reasoner for its processing. The **Human** class is seen as a subclass of **DeliveryMethod**, disjoint with siblings **Package**, **Vehicle**, see Figure 3.11. It is not too difficult to understand how the tree can assist in visualizing the hierarchy of a class such as **DeliveryMethod**. The class **DetectionMeans** is a means of finding an IED, but is not explicitly used in defining an IED. The **IEDIndicators** class is a depiction of objects that a

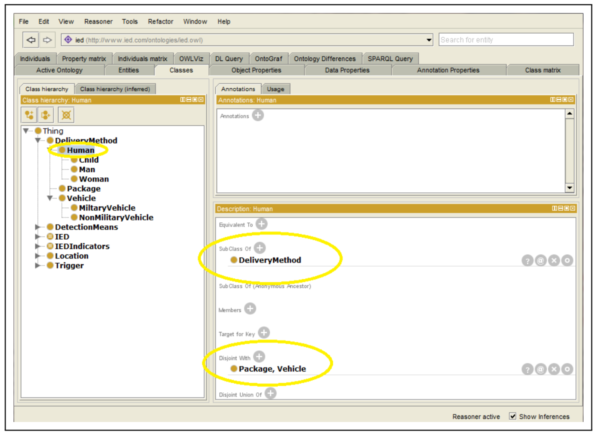

Figure 3.11: **Human** class shown with its disjoint sibling classes.

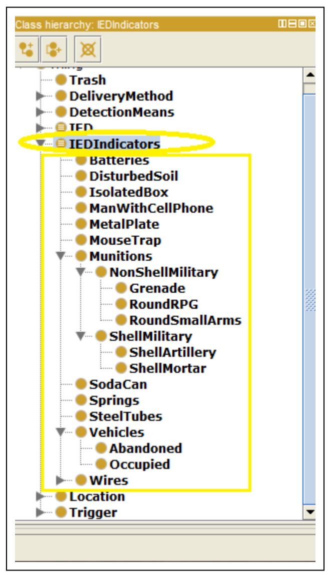

Figure 3.12: **IEDIndicators** class shown with with its subclasses contained in the yellow box.

user may be able to see when utilizing a UAS such as a Raven (see Figure 3.12). The **Trigger** class represents means to initiate an IED, see Figure 3.13. It would be plausible for a user of a UAS to see these types of objects while scanning an NAI for an IED. In **Trigger** class the subclass **CellPhone** is highlighted, notice that it is disjoint with its siblings.

3.5.4 Defining Properties in IED Ontology

Properties are what relate an individual to another individual in the ontology. An individual is an instance of a class. For example if there was a class “Dog” containing an individual named “Rascal” and he belongs to individual “John” from class “Owner,” “John” would share the

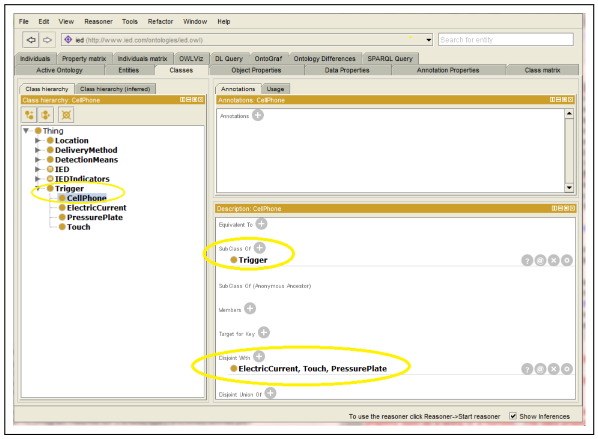

Figure 3.13: **Trigger** class with subclass **CellPhone** highlighted.

object property "hasPet" with individual "Rascal." In predicate calculus this takes on the form **hasPet**(**John**,**Rascal**). The same logic applies to defining an IED. There are four object properties that make up an IED ontology. The tutorial states "properties link individuals from the *domain* to individuals from the *range*" [43, 33]. Below are the four object properties defined in this ontology:

- **hasDeliveryMethod** Domain: **IED** Range: **DeliveryMethod**
- **hasIEDIndicators** Domain: **IED** Range: **IEDIndicators**
- **hasLocation** Domain: **IED** Range: **Location**
- **hasTrigger** Domain: **IED** Range: **Trigger**

The object property **hasDeliveryMethod** has three subproperties:

- **hasHumanDeliveryMethod** Domain: **IED** Range: **Human**
- **hasPackageDeliveryMethod** Domain: **IED** Range: **Package**
- **hasVehicleDeliveryMethod** Domain: **IED** Range: **Vehicle**

The **hasLocation** object property is selected in the Object Properties tab (see Figure 3.14) The **Functional** box checked implies that each individual in the **IED** class can be related to only one individual from **Location** class. The second class of properties in an OWL-Ontology are datatype properties. Datatype properties "...can be used to relate an individual to a concrete data value that may be typed or untyped" [43, 76]. The two data properties are **hasIEDIndicatorName** and **isConfirmed**, which are further summarized below:

- **hasIEDIndicatorName** Domain: **IEDIndicators** Range: **string**

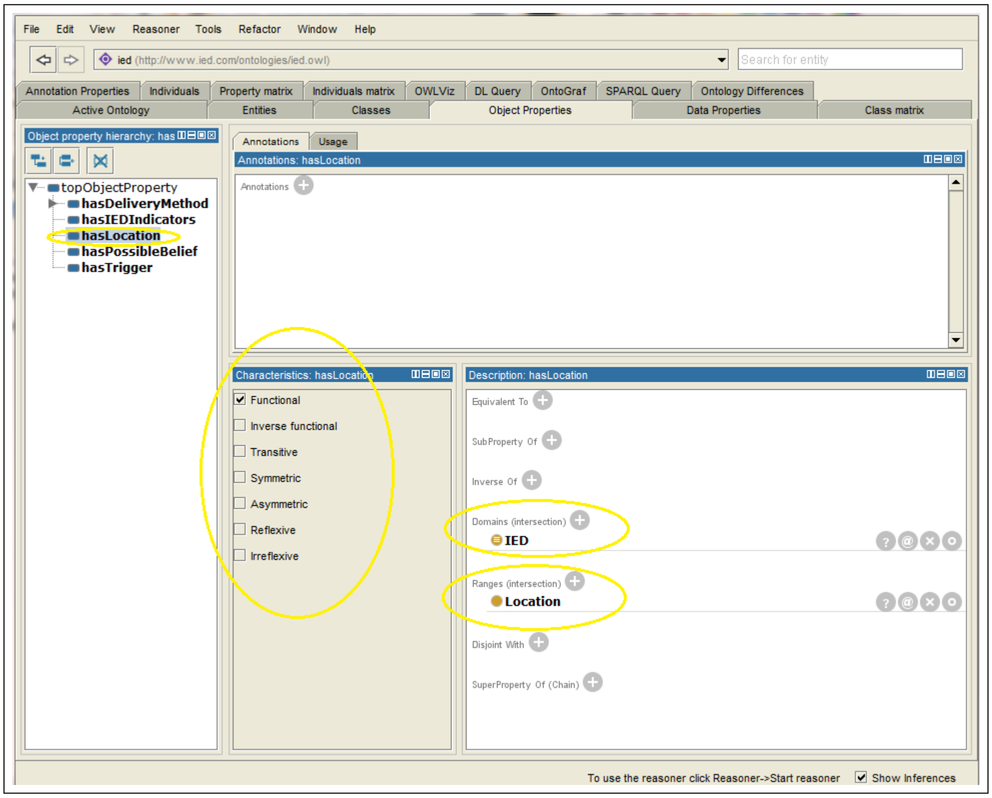

Figure 3.14: The **hasLocation** object property is selected with Domain **IED** and Range **Location**. The **Functional** box is checked as well.

- **isConfirmed** Domain: **IED** Range: **boolean**

The **hasIEDIndicatorName** range "string" will allow user to manually input some type of name for an **IEDIndicator** such as "dirt, iPhone, iron pipe." The **isConfirmed** allows the user to set if an **IED** indivdual is a confirmed IED (**isConfirmed** value equals "true") or not a confirmed IED (**isConfirmed** value equals "false").

3.5.5 Deeper Look in to IED Class

Now that the definition of classes and properties has been given, it is time to look further into **IED** class. The idea is to have the UAV sensor "see" objects ("clues") in its environment and the operator would take the "clues" and classify something as an IED. In order for something to be classified as being a member of the IED when using Protégé-Owl, the class has to be *necessary and sufficient* aka **Defined Class** [43, p. 55]. A necessary condition is one that needs to be present to be able to make the classification, but may need other conditions [43, 53]. A sufficient condition is one that by itself is enough to make the classification without needing other

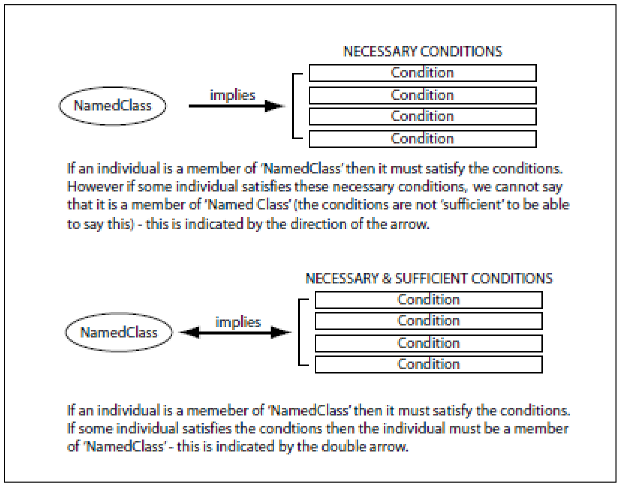

Figure 3.15: Necessary and Sufficient description. Taken from Pizza Tutorial [43, p. 56].

conditions to be true [43, 56]. See Figure 3.15 for an illustration of necessary and sufficient conditions. A class that is defined is indicated by white lines in the orange node circle [43, 57]. The necessary and sufficient criteria for **IED** class is seen in Figure 3.16. The word *some* represents an existential restriction. This says that there is a relationship between individuals of IEDs that has "at least one" relationship along the property (for example) "**hasDeliveryMethod**" to an individual that is a member of the class "**DeliveryMethod**" [43, 59]. The "and" represents an intersection where all of the object properties overlap. Next, we will look at the three different types of IEDs based on how they are delivered: Package (**PackageIED** class), Suicide/Human (**SBIED** class), and Vehicular (**VBIED** class).

The **PackageIED** class is a subclass of **hasPackageDeliveryMethod** and **IED** and it inherits the same four characteristics of an IED based on being a subclass of **IED**. The **PackageIED** class represents delivery by some package, one that is not a suicide bomber (SBIED) or vehicular (VBIED). One can see that it is also disjoint with its siblings **SBIED** and **VBIED** (see Figure 3.17). The **SBIED** class is a subclass of **IED** and **hasHumanDeliveryMethod some Human**, and like its sibling **PackageIED** it inherits the four properties that make an IED an IED. It is also disjoint with **VBIED** and **PackageIED** (see Figure 3.18). The purpose of **SBIED** is to represent characteristics of a suicide-borne IED. A suicide borne IED delivery could be performed by a child (**ChildSBIED**), Woman (**FemaleSBIED**) or a man (**MaleSBIED**). Since **ChildSBIED** is a subclass of **SBIED** it inherits the properties of **SBIED**, which is defined in the middle of the "Description: ChildSBIED" box labeled

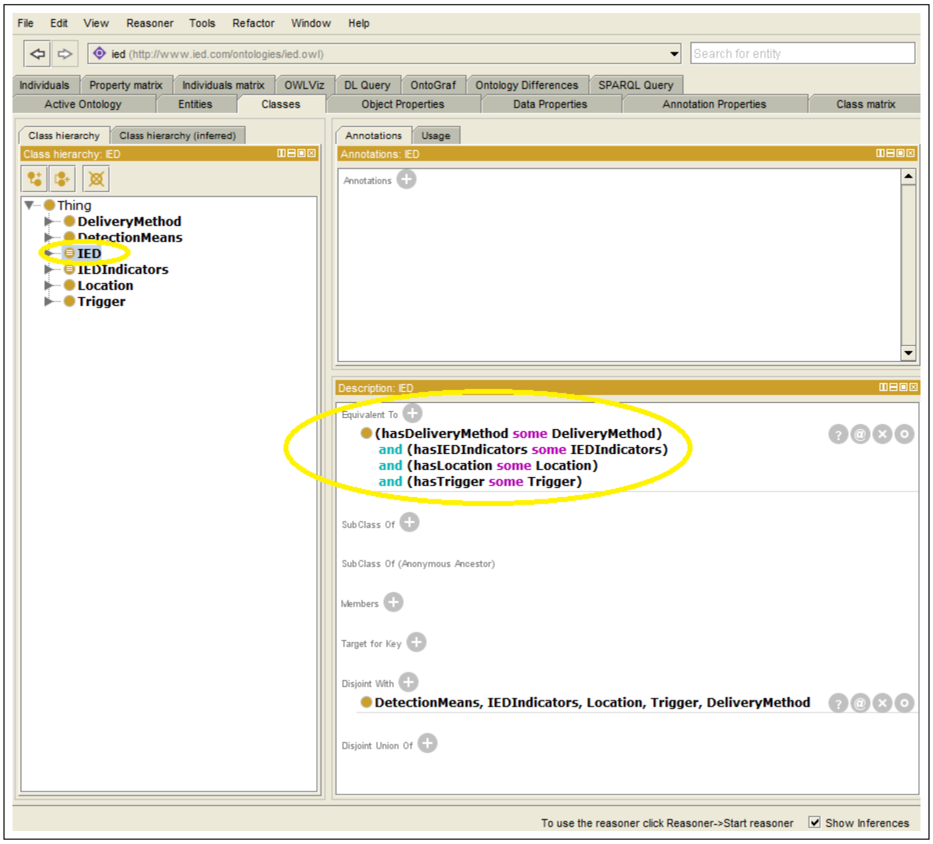

Figure 3.16: The "Equivalent To" box is highlighted showing what must be true in order for an individual to be a member of the **IED** class.

"Sub Class of (Anonymous Ancestor)." The property that explicitly defines **ChildSBIED** is **hasHumanDeliveryMethod** and qualifies the delivery method as a child with the "some child" clause. It is also disjoint with its siblings **MaleSBIED** and **FemaleSBIED**. The classes **MaleSBIED** and **FemaleSBIED** are similar to **ChildSBIED** with **hasHumanDeliveryMethod** qualified as Female and Male, respectively. The **VBIED** class is a subclass of **IED** and **hasVehicleDelivery** and like its siblings **PackageIED** and **SBIED** it inherits the four properties that make an IED an IED. It is also disjoint with **SBIED** and **PackageIED** (see Figure 3.19). The purpose of **VBIED** is to represent characteristics of a vehicle-borne IED. The class **ConfirmedIED** is to be used when an operator sees characteristics of an IED (via UAV) and calls EOD, who then "confirms" that it is a real IED. It inherits the properties of **IED** and has the value **isConfirmed** with a "true" value. The **ConfirmedIED** is disjoint with its siblings **SBIED**, **VBIED**, and **PossibleIED** (see Figure 3.20). If the IED is not confirmed, it will then be classified as a "PossibleIED," which is discussed next.

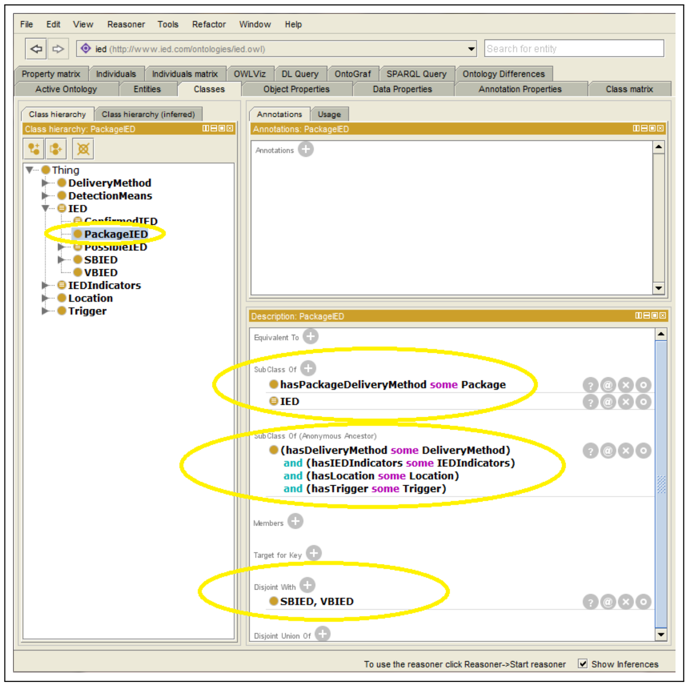

Figure 3.17: **PackageIED** shown.

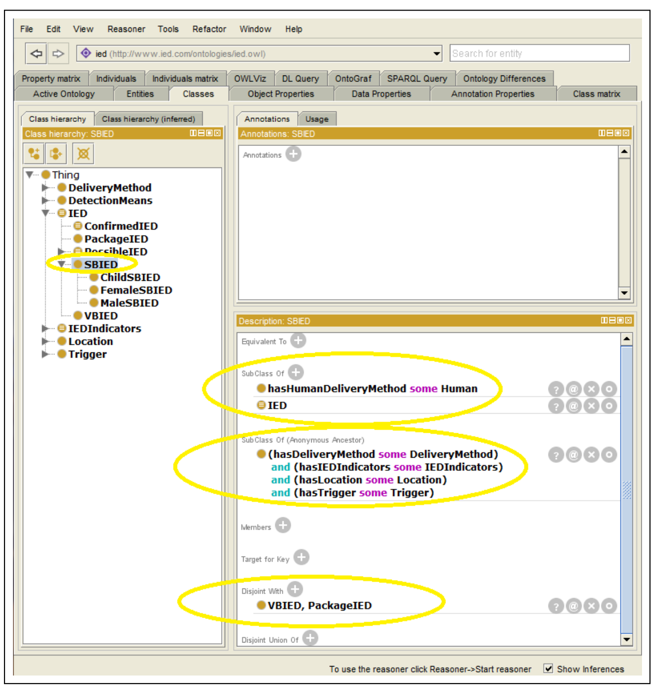

Figure 3.18: **SBIED** shown.

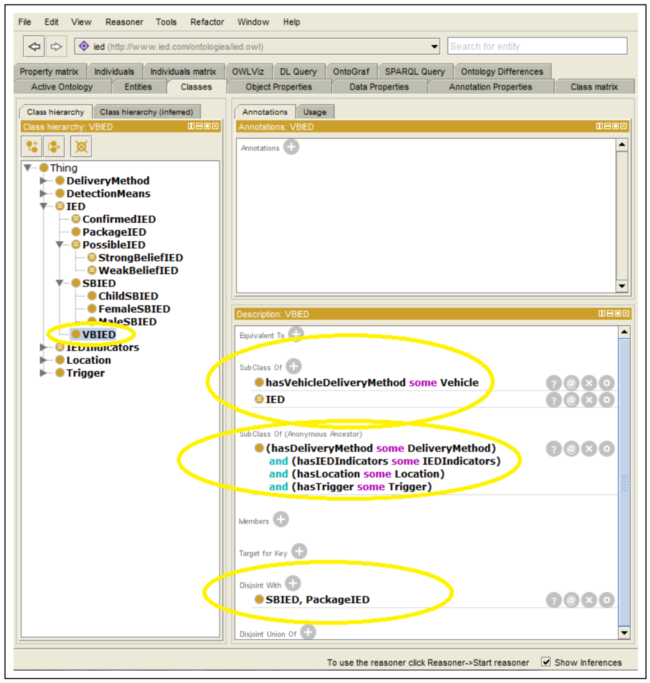

Figure 3.19: **`VBIED`** shown. A vehicle borne IED delivery happens by a vehicle via **`hasVehicleDeliveryMethod`**.

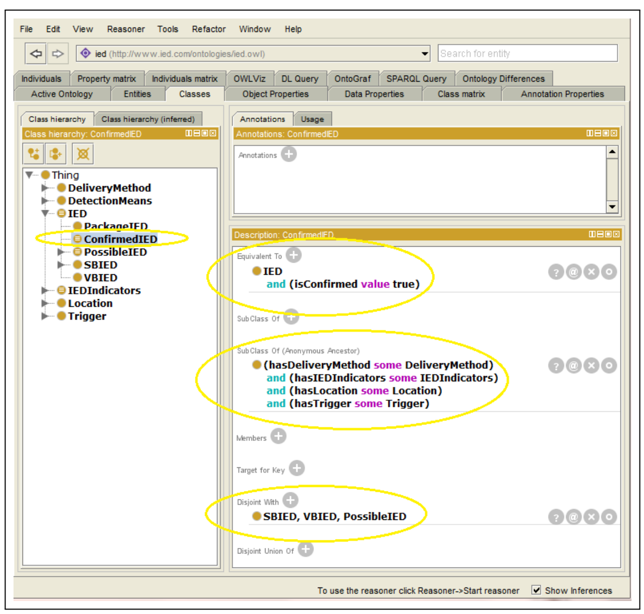

Figure 3.20: **`ConfirmedIED`** shown.

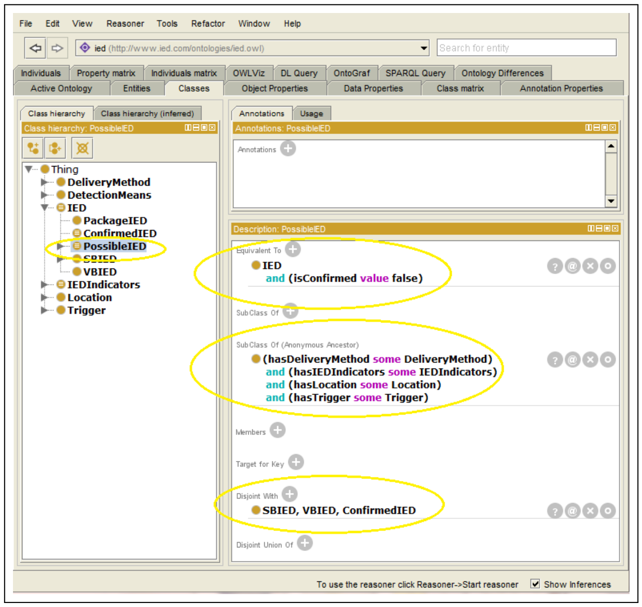

Figure 3.21: **PossibleIED** shown.

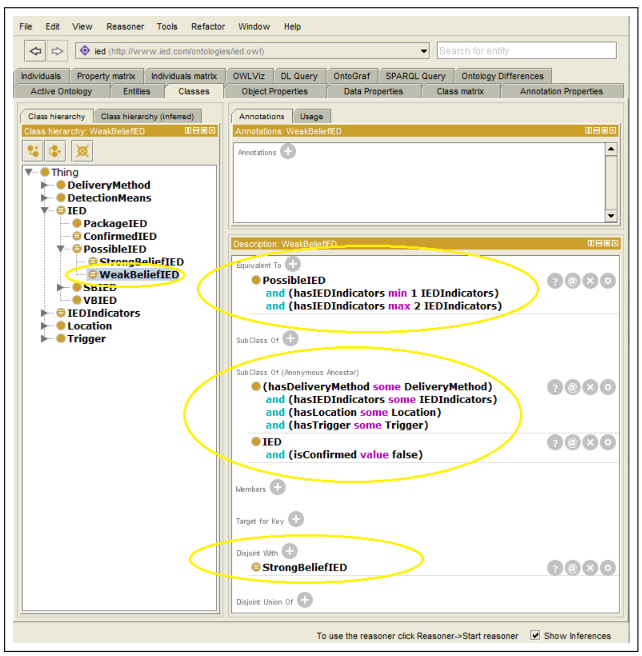

Figure 3.22: **WeakBeliefIED** shown.

The **PossibleIED** is a subclass of **IED** and has the property **isConfirmed** with a "true" value. In theory if an IED is not a "ConfirmedIED" it will always be classified as a "PossibleIED" (see Figure 3.21). A **WeakBeliefIED** inherits all of the properties of a type **PossibleIED** with the exception of its own specific requirement in "Equivalent to" box (circled on top portion of "Description" window, see Figure 3.22). A **`WeakBeliefIED`** is a type of IED that has a minimum of 1 IED indicator and a maximum of 2 IED indicators. It is using a new restriction called the universal restriction *only*. The universal restriction states "*if* a relationship exists for the property then it must be to individuals that are members of a specific class" [43, p. 60]. The goal is to create an individual with a minimum of 1 and maximum 2 "IEDIndicators" and have the reasoner classify it as a "WeakBeliefIED." The minimum and maximum are a type of cardinality restriction. The tutorial states: "A Cardinality Restriction specifies the *exact* number of P relationships that an individual must participate in" [43, p. 73]. The assumption is that you would need a few indicators such as loose soil and wires together at some specific location along a route of travel to have a UAS operator to possibly take a closer look at what they are scanning on the ground. If more than two indicators are found, the reasoner will classify the object as a "StrongBeliefIED," which is described next. A **`StrongBeliefIED`** inherits all characteristics of a **`PossibleIED`** with the addition of a third "IEDIndicator." Therefore in the "Equivalent To" box the addition of object property **`hasIEDIndicators min 3 IEDIndicators`** is necessary. The idea is to show that an operator that sees "loose soil," "box," and "wires" would probably think that there is a higher suspicion of being a true IED versus only seeing two or fewer indicators. The properties of a **`StrongBeliefIED`** can be found circled in the "Description: StrongBeliefIED" box, see Figure 3.23.

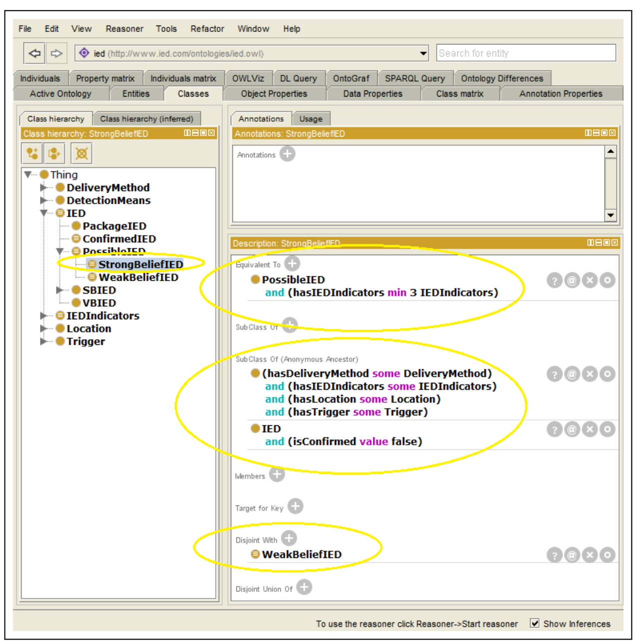

Figure 3.23: **`StrongBeliefIED`** shown.

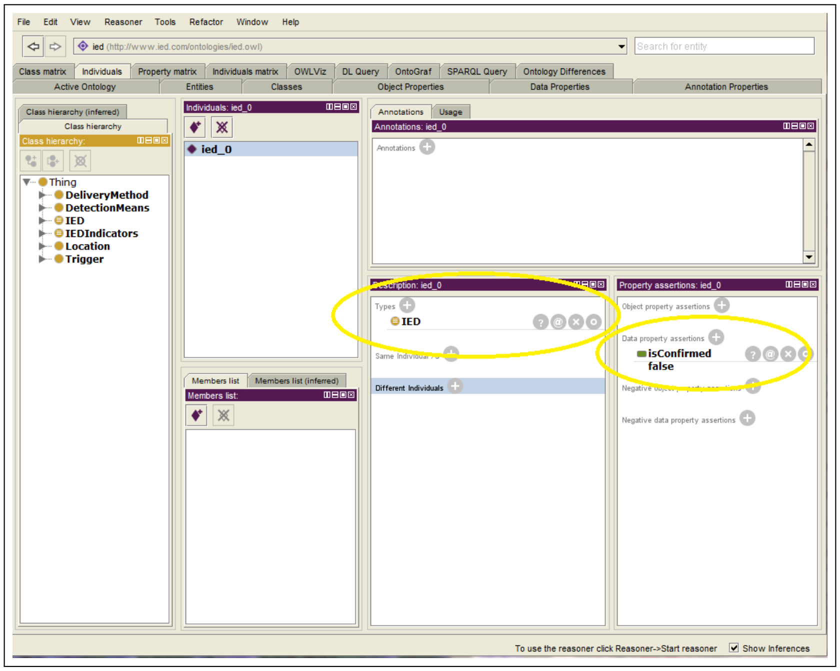

Figure 3.24: **`ied_0`** individual shown.

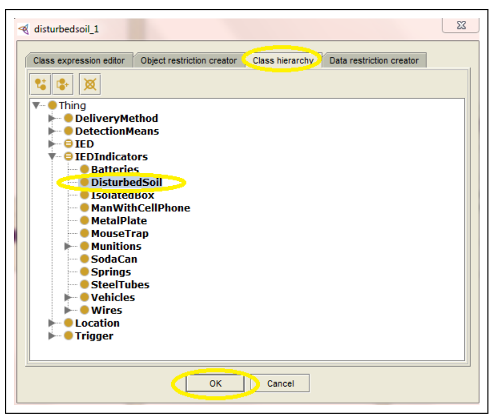

Figure 3.25: **`disturbedsoil_1`** individual being assigned type **`DisturbedSoil`**.

3.5.6 Defining Individuals in IED Ontology

The study will now look at creating individuals in the ontology that will allow the reasoner to show if the created individual can be classified as a "PossibleIED," "StrongBeliefIED," "Weak-BeliefIED," or "ConfirmedIED." Individuals can be created in the "Individuals" tab. All individuals in Protégé are identified with a purple diamond and are written in lowercase. In order for the reasoner to classify (infer) an individual as being a member of a certain class, the class of which it is a member needs to be a "defined" class, otherwise the reasoner will not properly classify (infer) its relationship to a particular class. First, the individual **ied_0** is created, see Figure 3.24. It is given the type **IED** and Data property assertion **isConfirmed** with value "false." This represents an object that has been detected but no indicators have been identified. The second IED individual will have with it two IED Indicators: "disturbed soil" and "wires." These indicators will need to be created individually so we can assign them as object properties to individual **ied_1**. First IED Indicator is named **disturbedsoil_1**, assigned class type **DisturbedSoil** and given **hasIEDIndicatorName** "dirt" string. To assign the type **IED** click on "Types" "+" sign and select "Class Hierarchy" and select "ok." (see Figure 3.25). Next, assign data property assertion type string "dirt" to **disturbedsoil_1** indicator. See Figure 3.26 for assigning "dirt" to indicator **disturbedsoil_1**. One could change "dirt" to "dog" and the reasoner would not care, assigning "dirt" makes it easier for the user to follow along. The second IED Indicator is named **wires_1** it is assigned the **IEDIndicator**

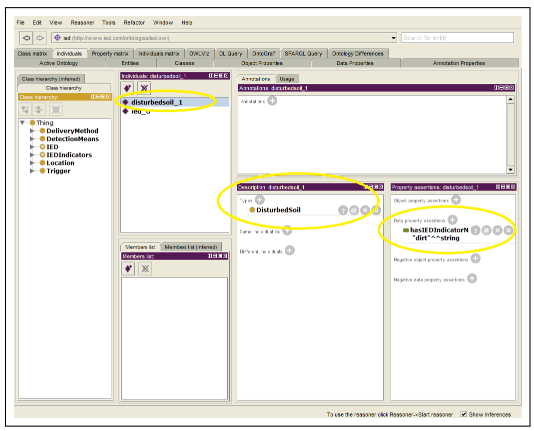

Figure 3.26: **disturbedsoil_1** indicator being assigned string "dirt" shown.

subclass **Wire** and given the data property assertion string "wire" (see Figure 3.27). The second IED individual, **ied_1** is like **ied_0** except the individuals **disturbedsoil_1** and **wires_1** are assigned as "object property assertions," and since this is not a "confirmedIED" it is assigned data property **isConfirmed** with value "false." (see Figure 3.28). To assign the IED Indicators select "Object property assertions" underneath "property assertions window" and click on "+" sign. Next assign the two indicators to **ied_1** (see Figure 3.29). The third and final indicator is defined with object property **hasIEDIndicator** with type string and individual assigned the name as **isolatedbox_1** to represent a box alongside the road. There is no change in how you assign it a type and data property assertion. See Figure 3.30 for **isolatedbox_1** individual screenshot. The third IED individual is **ied_2** it has 3 IED

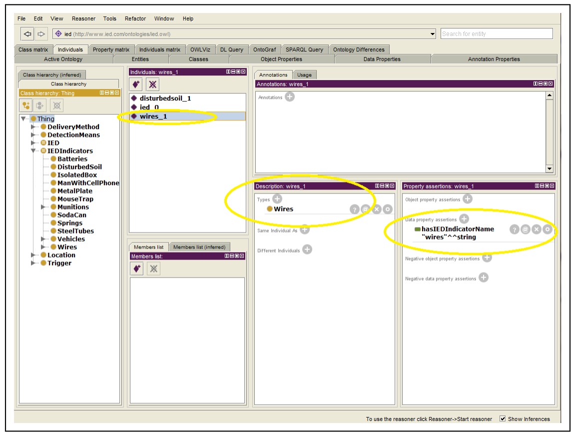

Figure 3.27: **`wires_1`** individual shown.

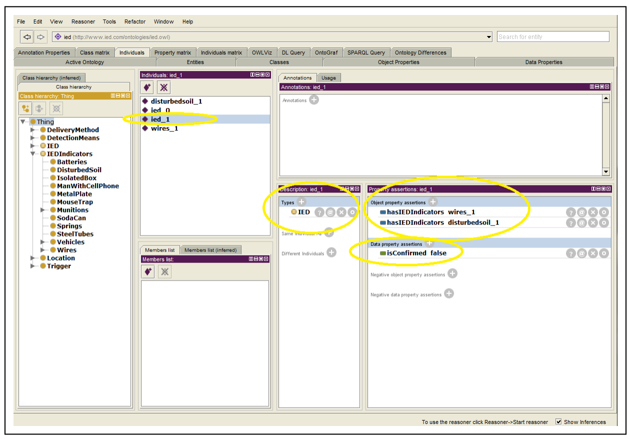

Figure 3.28: **`ied_1`** individual shown.

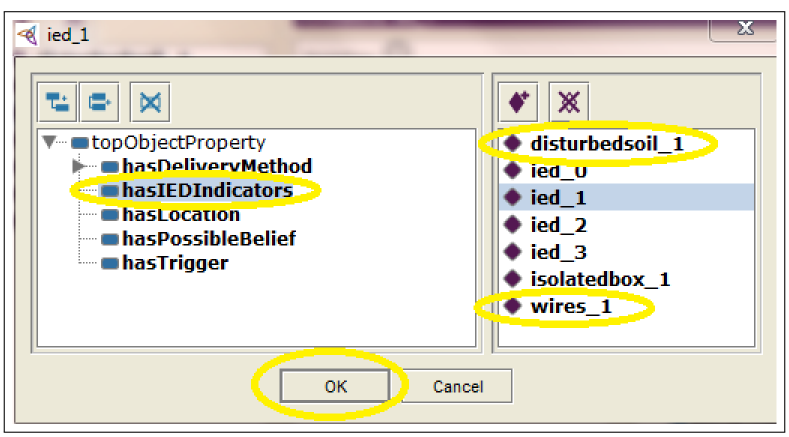

Figure 3.29: Assigning IED Indicators to **`ied_1`**.

Indicators and “isConfirmed” value set to “false” (see Figure 3.31). This is an individual that an operator has been able to associate three characteristics of this “Possible IED.”

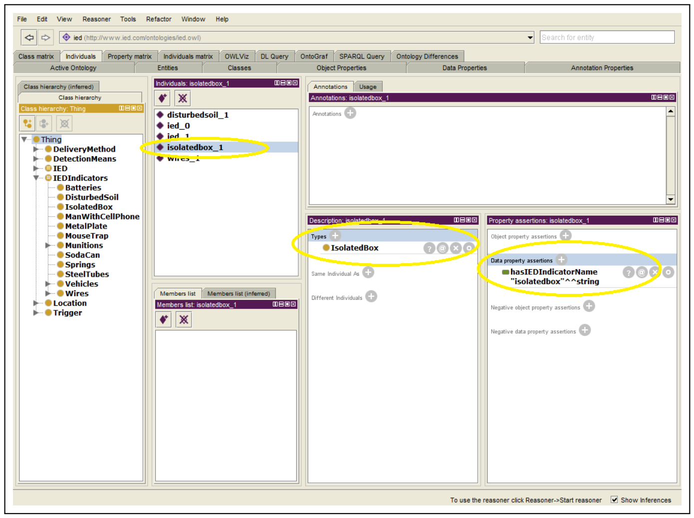

Figure 3.30: **isolatedbox_1** individual shown.

Since it is not confirmed yet, the data property **isConfirmed** is set to “false.” The fourth and

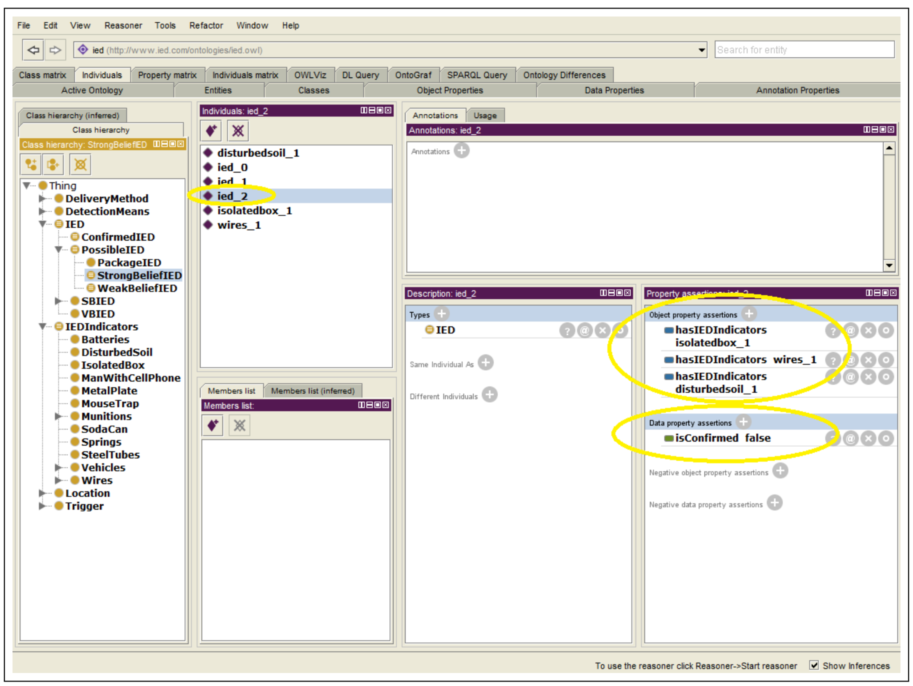

Figure 3.31: **ied_2** individual shown.

final IED individual is **ied_3**, which is just like **ied_2** except this IED has been confirmed by EOD to be a legitimate IED, therefore the **isConfirmed** data property assumption is set to "true" (see Figure 3.32).

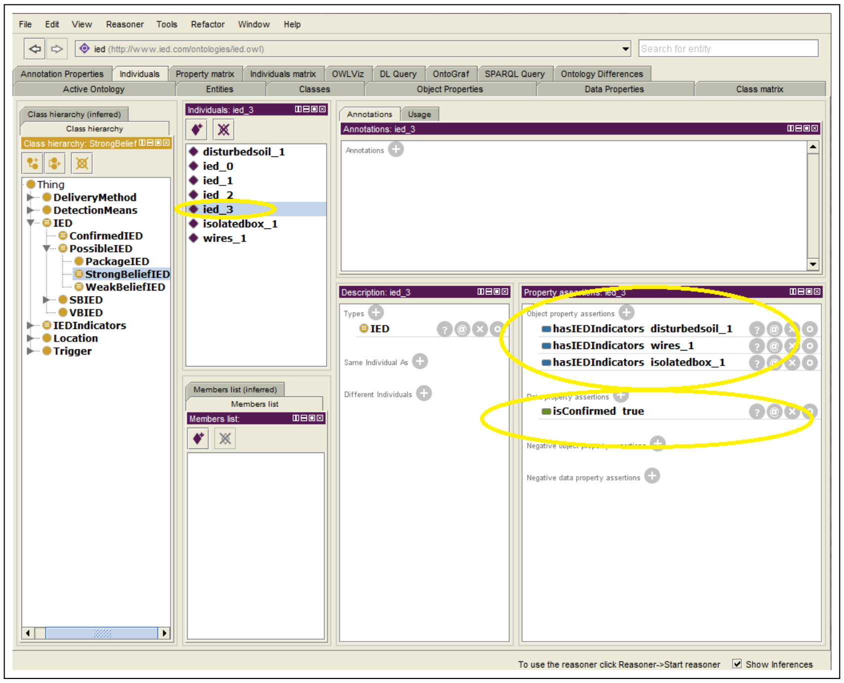

Figure 3.32: **ied_3** individual shown.

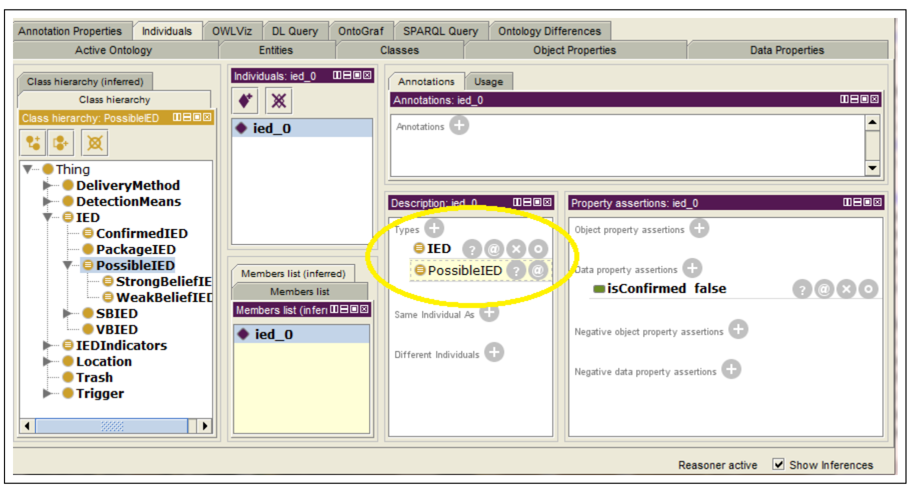

Figure 3.33: Reasoner classifies object "Possible IED." based on its object and data properties.

3.5.7 Classifying an IED Using the Reasoner

The experiment will now model the behavior of a UAS's ability to make a decision based on the information that it receives from its sensors. In this experiment, the sensor (UAV) is collecting data that falls into the categories of "IED Indicators." Pieces of information that an operator may or may not be able to see using the UAS is represented as individuals. The reasoner is used to establish the level of confidence in what is seen based on the amount of information the operator is able to see using his/her UAV. The UAV is flying looking for threats along the way. The operator detects an object (called **ied_0**), however, he/she cannot identify what it is. Because **ied_0** does not have any indicators the reasoner should and does classify this object as a "PossibleIED" (see Figure 3.33). As the operator continues to look at the object

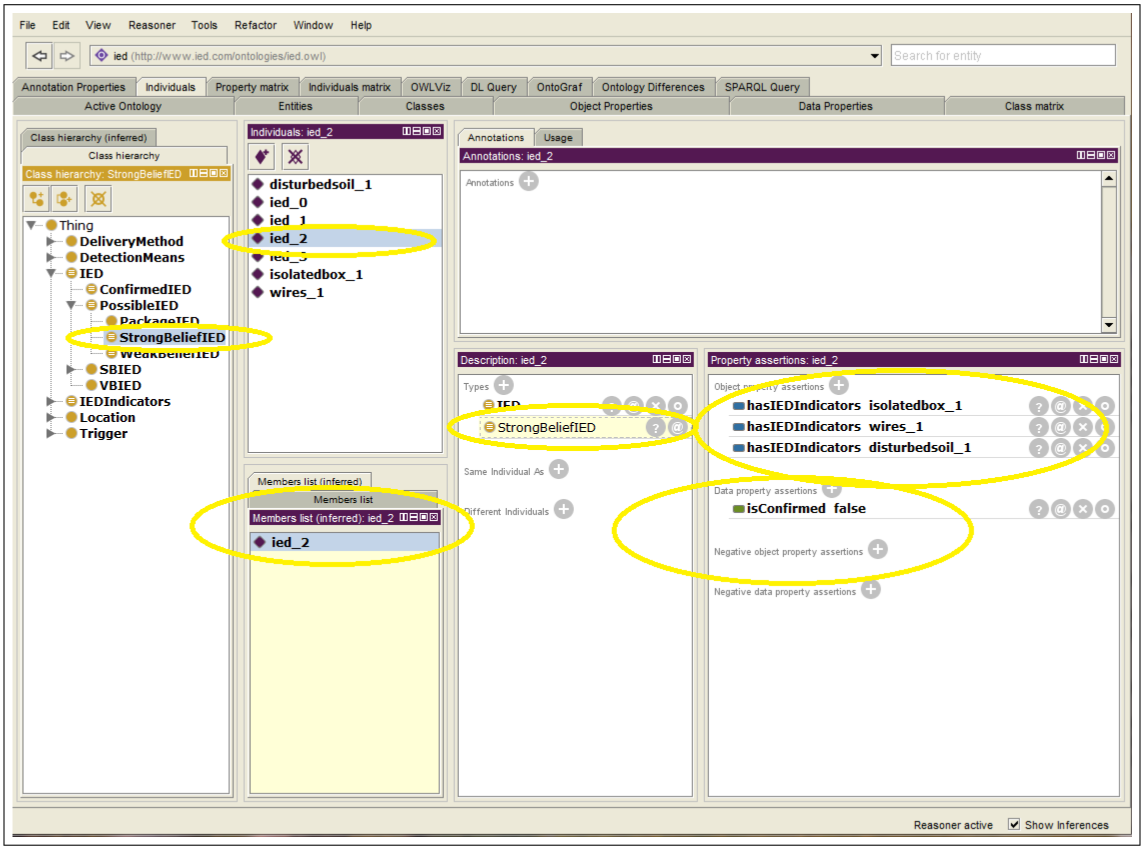

Figure 3.34: Reasoner classifies object as “Strong Belief IED.” based on properties associated with **ied_2**.

they see IED indicators such as disturbed soil, wires, and now a box (making the user feel more confident about what is seen) causing the reasoner to classify this object (now called **ied_2**) as a “Strong Belief IED” (see Figure 3.34) since there are three indicators. The convoy commander cordons off the area and calls for the EOD to investigate this strong belief, possible IED. Once EOD arrives, having looked at the same characteristics that the operator has, confirms that object (now called **ied_3** is a “real IED” and classifies it as a “Confirmed IED” (see Figure 3.35).This device is neutralized by EOD and a highly possible catastrophic kill is prevented.

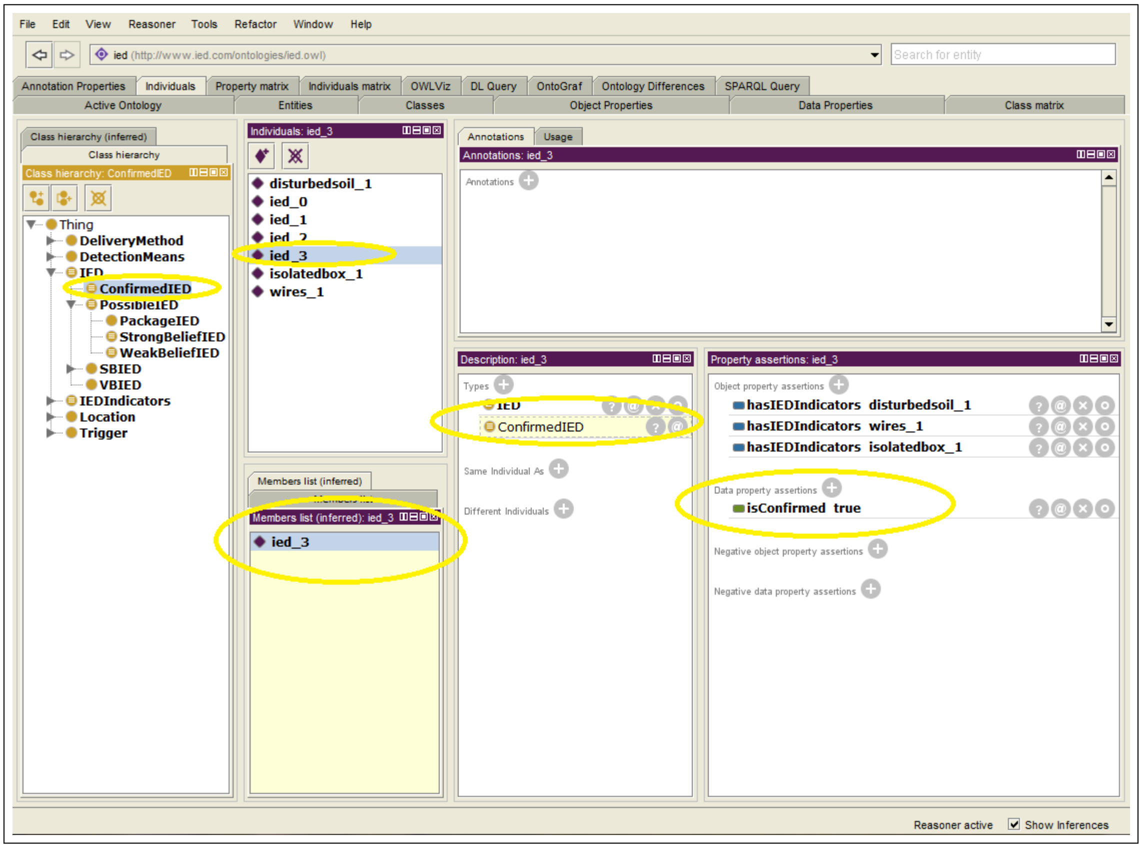

Figure 3.35: The “possible, strong belief IED” has been confirmed an IED by explosive ordnance disposal (EOD).

3.6 DESIGN OF EXPERIMENTS

The goal of Design of Experiments is to see if the effects of training on mission success can be shown in COMBATXXI utilizing the HTNs and UAV Sensor model. The mission objective is to arrive at Outpost X-Ray safely (as mentioned in section 3.2). The only true IED (located in NAI #2) detonates with probability 0.80 and will always kill the second vehicle in the convoy when detonated, the other three NAIs each have a single non-IED object. The mission will be over if: (1) Convoy reaches final destination without hitting IED (Mission Complete or MC) or (2) Convoy hits IED (Mission Failure). Success (MC) can only happen if IED is detected and defeated (UAS used effectively) or IED does not go off (probability 0.20) given IED is undetected (worse case). The UAV sensor [40] has a likelihood of detecting the object with probability 0.50 if the range from the sensor to the IED (r) is 110 meters and probability 0 if r is 220 meters or greater. Time to clear an IED is 60 minutes.

The factors affecting mission success are False Positive (α) and False Negative (β), with levels ranging from 0.0 to 1.0. Experiments run at six values for each (0.0, 0.2, 0.4, 0.6 0.8, 1.0), for a total of 36 design points. Each design point is run 100 times for a total of 3600 observations (n). Below are the definitions for False Positive and False Negative:

- If False Positive (α) 0.0 then UAS ALWAYS identifies a non-IED object as a non-IED.
- If False Positive (α) 1.0 then UAS ALWAYS identifies non-IED as an IED.
- If False Negative (β) 0.0 then UAS ALWAYS correctly identifies IED as an IED.
- If False Negative (β) 1.0 then UAS ALWAYS incorrectly identifies IED as a non-IED.

Best case is when alpha and beta are 0.0 and 0.0. This represents an operator that both correctly classifies an object as being an IED or something that is truly not an IED (trash), reducing risk (less time exposed), and saving resources (UXO clearing team not called).

Worst case is when alpha and beta are 1.0 and 1.0. This represents an operator who always incorrectly detects an object as an IED and unnecessarily wastes resources to "clear" a non-IED, adding more time to convoy being exposed outside the wire, while at the same time always misidentifying real IEDs, leading to a high failure rate.

Metrics to measure the effects of alpha and beta are:

- Mean Mission Success Rate
- Mean Time to Complete Mission | Mission Complete
- Risk = 1/MC% x Mean Time to Complete Mission | Mission Complete

3.6.1 Summary

The concept of using an ontology and implementing it in order to demonstrate a different approach in measuring the sensing ability of a UAS is discussed. The DOE will determine if it is possible to model the positive and negative effects of being trained and untrained (changing values of alpha and beta) given the UAV sensor model detects the object. In the next chapter, the research will perform an analysis on data to be collected.

CHAPTER 4: ANALYSIS

4.1 INTRODUCTION

We demonstrate the utility of our approach through analysis of the proof of concept scenarios. The findings show how the scenario results are sensitive to changing parameters False Positive (α) and False Negative (β) in our sensor model. This proof of concept scenario would be useful to a decision maker interested in evaluating the potential impact of training on the effective use of UAVs.

4.2 DESIGN OF EXPERIMENTS ANALYSIS IN COMBATXXI

The analysis will investigate the effects of α and β at six levels each (0.0, 0.2, 0.4, 0.6, 0.8, 1.0), mentioned earlier in chapter 3 "Methodology" The goal is to identify what combinations of alpha and beta give us the highest Mean Mission Success Rate, lowest Mean Time to Complete Mission | Mission Success, and to help quantify Risk.

4.2.1 Mean Mission Success Rate

This section is about measuring the mean success rate for the 3600 observations. A heat map is utilized to communicate results. Boxplots help to show noise and meaning to support the findings on α's and β's effects on Mission Success Rate. Looking at Figure 4.1 we can see that the highest completion rate is 0.84 and occurs when β is 0.0, and is independent of α

False Negative (β) Parameter	False Positive (α) Parameter: 0.0	0.2	0.4	0.6	0.8	1.0
0.0	0.84	0.84	0.84	0.84	0.84	0.84
0.2	0.74	0.74	0.74	0.74	0.74	0.74
0.4	0.57	0.57	0.57	0.57	0.57	0.57
0.6	0.46	0.46	0.46	0.46	0.46	0.46
0.8	0.29	0.30	0.29	0.29	0.29	0.29
1.0	0.20	0.19	0.19	0.19	0.19	0.19

Mean Mission Success Rate per 100 trials

Figure 4.1: Heat Map of Mission Success Rate versus parameters False Positive (α) and False Negative (β). Notice that the Mission Success Rate does not change as alpha increases from 0.0 to 1.0, which demonstrates that the False Positive parameter α does not have an effect on mission success. Mission failures occured even with β at zero because on some trials the UAV never sees the IED, and as a result the IED was not identified.

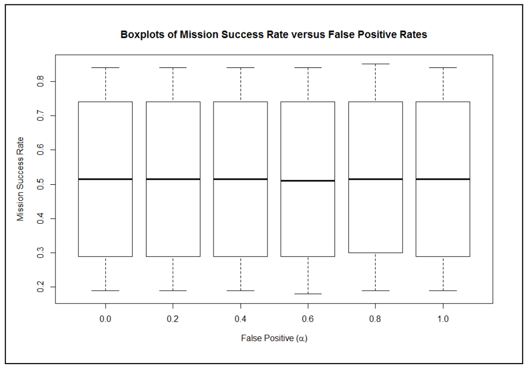

Figure 4.2: Boxplot of Mission Success Rate versus False Positive Rates. Notice that the Mission Success Rate does not change as alpha increases from 0.0 to 1.0. This shows that α does not have an effect on mission success since False Positive represents the identification of non-IED objects.

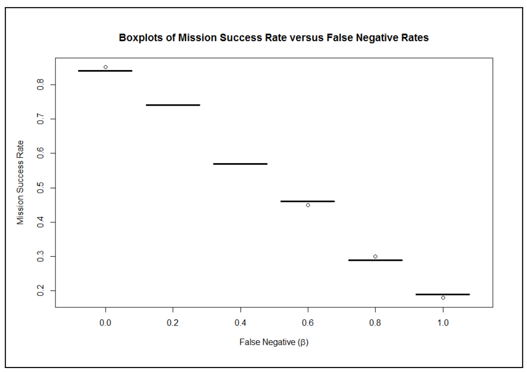

Figure 4.3: Boxplot of Mission Success Rate versus False Negative Rate (β). Notice that the Mission Success Rate decreases as β increases from 0.0 to 1.0. There is no variance in the success rate either, which is why the boxes appear as lines instead of boxes per earlier boxplots when looking at Mission Success versus False Positive Rate. This shows that β has a great effect on mission success rate versus α.

levels. The lowest average of completing mission is approximately 0.20 and 0.19 when β is 1.0, independent of α levels. The decrease in mean mission success as β increases is caused by a higher probability of incorrectly identifying the IED in NAI #2 as a non-IED. Since the IED has a probability of 0.8 of detonating, that is why the success rate is 0.20 when β equals 1.0. As the False Positive value increases from left to right the success rate generally remains constant. It is only as we increase β that there is a difference in mission success rate, concluding that β has a strong influence on the likelihood of completing the mission. Looking at the first boxplot (see Figure 4.2), one can see that the False Positive (α) does not have an effect on the mission success rate since the boxplots share the same dimensions, respectively. The Mission Success

False Negative (β) Parameter	False Positive (α) Parameter: 0.0	0.2	0.4	0.6	0.8	1.0
0.0	125.00	145.00	166.43	196.43	216.43	241.43
0.2	123.19	141.02	163.73	192.92	214.81	239.13
0.4	119.06	138.01	158.01	185.38	205.38	227.48
0.6	116.00	134.26	156.43	183.82	204.69	225.56
0.8	101.61	124.43	136.78	165.74	184.36	209.19
1.0	69.44	91.70	113.80	139.06	161.17	186.43

Mean Time to Complete Mission | Mission Success

Figure 4.4: Heat Map of Mean Time to Complete Mission | Mission Success for different values of False Positive (α) and False Negative (β). Mean time to complete the mission takes the longest when α is 1.0 and β equals 0.0. Mean time for a successful mission increases as α increases and decreases as β increases.

Rate is very close or right above 50% as α increases from 0.0 to 1.0. Continuing on with the second boxplot, the False Negative Rate (β), mission success decreases as β increases from 0.0 to 1.0 (see Figure 4.3).

This visual helps to show that in the model that False Negative Rate (β) has a strong effect on mission success.

4.2.2 Mean Time To Complete | Mission Success

Now the study will look at the mean time to complete the mission given mission success occurs. Looking at the heat map in Figure 4.4 it appears that the mean time to complete mission given mission success is influenced more on α than β. Mean time to complete mission is highest at 241.43 minutes when $\alpha = 1.0$ and $\beta = 0.0$. This makes sense; the operator will always identify a non-IED as an IED when $\alpha = 1.0$. The lowest mean time to complete mission of 69.44 minutes is at the opposite corner when $\alpha = 0.0$ and $\beta = 1.0$. This number can be misleading because in this case, the UAV operator will always identify a piece of trash as trash ($\alpha = 0.0$), but when β is equal to 1.0, the operator will always incorrectly identify an IED as a non-IED. Therefore in this case the convoy will never stop to handle a potential threat. The only reason there is any success in this case is because there is a 0.20 chance the IED will not detonate and it is in these "lucky" cases that the mission is completed. Now examining the Mission Complete Time boxplots (see Figure 4.5) it appears that the average time to complete the mission does increase as False Positive Rate (α) increases from 0.0 to 1.0.; this makes sense in our model given that we stated earlier that if α is 0.0 it would always identify a non-IED object as a non-IED. Also when α is 1.0 the UAS operator always identifies non-IED as an IED, leading to additional time spent on a mission and exposing onself to more harm. Examining Figure 4.6 it appears that mission completion time does not vary as much and remains constant (at β value increases), proving that in the model, the False Negative Rate (β) does not have as much influence for mean

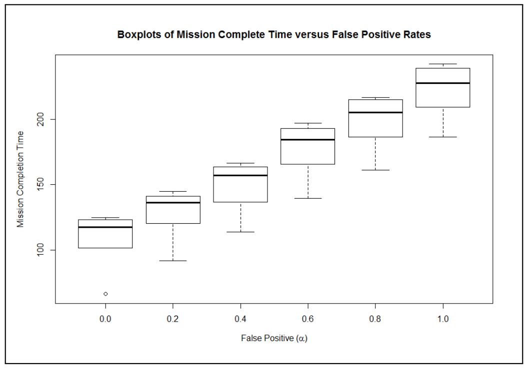

Figure 4.5: Boxplot of Mission Completion Time versus False Positive Rates. Notice that median time to complete mission increases as the False Postive Rate (α) increases. There appears to be a linear relationship; the variance for each value of α appears to be equal to one another as well. This means that α will always have an effect on amount of time it completes the mission.

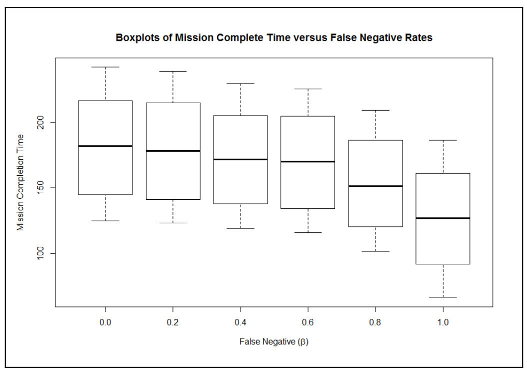

Figure 4.6: Boxplot of Mission Complete Time versus False Negative Rate (β). The False Negative parameter β does not have as much of an effect on average time it takes the convoy to complete the mission. Notice that the Mission Completion Time is not heavily influenced on the change in β from 0.0 to 1.0. A time of 150 minutes could be found for each β value. Only when the value of 0.8 is reached is there a drop in median time of completing the mission. This is because the observer has a higher probability of incorrectly identifying the IED as a non-IED, resulting in fewer times to successfully complete the mission and have an observed time.

mission completion time as the False Positive Rate (α).

4.2.3 Measuring Risk

Risk is something all commanders must take when utilizing the U.S. Army's most valuable asset: soldiers. As a means to assess risk, we can combine the considerations of mission success rate and completion time. We task risk to be:

False Positive (α) Parameter

False Negative (β) Parameter	0.0	0.2	0.4	0.6	0.8	1.0
0.0	148.81	172.62	198.13	233.84	257.65	287.41
0.2	166.47	190.57	221.25	260.70	290.28	323.15
0.4	208.88	242.12	277.21	325.22	360.31	399.09
0.6	252.17	291.86	340.07	399.61	444.98	490.35
0.8	350.37	414.77	471.64	571.52	635.73	721.34
1.0	347.20	482.63	598.96	731.91	848.26	981.21

Risk = 1 / MC% * Mean Time To Complete Mission | Mission Success

Figure 4.7: Heat Map of Risk for different values of False Positive (α) and False Negative (β). Lowest risk takes place at α and β being 0.0, 0.0. Given the IED is detected, the user has capability to correctly identify the IED and does not waste time clearing non-IEDs. When α, β is 1.0, 1.0, risk is highest (less likely to correctly identify IED and more likely to incorrectly identify non IEDs as being IEDs).

$$Risk = 1/MC\%x(MeanTimetoCompleteMission|MissionComplete) \tag{4.1}$$

Figure 4.7 shows risk as a function of α and β. It is the most interesting of the three heat maps. This measure of risk attempts to combine the risk of not identifying the IED with the inherent risk associated with being out in the open for an extended amount of time. Figure 4.7 confirms that this characterization of risk captures both aspects of the risk encountered by convoys. We expected the most dangerous situation to be when α is 1.0 and β is 1.0, when the operator always misidentifies objects as being real IEDs and never identifies the real IEDs as a real IED, this is what we can see in the heat map. At the opposite corner when α and β are 0, respectively, risk measures at 148.81. This is the best case scenario because an IED is always classified as an IED and non-IEDs (trash) are always identified as trash, minimizing the amount of time a convoy must spend on the route outside of "friendly" territory while still reducing the vulnerability to damage from IEDs.

4.3 SUMMARY

The results above show that it is possible that a UAS (like the Raven) may improve deployable force protection capabilities if the assumptions hold true in the real world. It is possible to measure success and time to complete a mission with different combinations of α and β. The heat maps provided valuable insight on the ramifications of each α and β level. The boxplots gave reassurance in the validity of the heat maps by showing differences the parameters do or do not have. Risk gave further understanding into the importance of correct versus incorrect identification of objects, and the effects of wasting time clearing non-IED threats. Next, the study will give a conclusion of using the new STA model and the idea of combining KR through an ontology made using special software.

CHAPTER 5:
CONCLUSION

5.1 OUTCOMES

We demonstrated the application of the new STA model in a tactical convoy scenario in COMBATXXI. We showed the ability to model a UAS in a working environment is fundamental to a study concerning tactical convoy operations. A design of experiments and post analysis quantifies the sensitivity of the measures of effectiveness of success to conditions contributing to variability in UAS performance.

5.1.1 What Does This Mean?

Behaviors for UASs can be designed and developed using Behavior Studio Interactive Design Environment. HTNs were created to replicate the thought and action process of a convoy as it travels down a path wary of threats such as IEDs, with respect to the use of UAVs. The simulation allows for some analysis to determine if tactical convoy operations using a deployable force protection asset (Raven UAS) could be studied in COMBATXXI using the UAV sensor model, and associated dynamic behaviors without the reliance of nonadaptive scripting and compound orders currently in use. Factors such as False Positive and False Negative allows us to see how an operator's skill level can affect the outcome of a tactical convoy operations in SWA. It was found that a convoy experienced less risk and obtained a higher frequency of completing the mission when they could increase the likelihood of identifying an IED object as an IED and decreasing the likelihood of misidentifying non-IEDs.

5.1.2 Are There Any Advantages to UAS?

Given the constraints, conditions, and limitations of the sensor model and COMBATXXI, it was shown that utilizing the UAS gave a higher likelihood of success. We also showed that qualitative factors such as risk could be be controlled with the use of a UAS.

5.1.3 Have We Modeled UAS properly?

Due to the dynamics of electro-optics (EO) sensors, the human eye and thought process, it is naive to say the Raven UAS was modeled properly, in this study. This study demonstrates that it is possible to model the complexities of a UAS in a complex simulation such as COMBATXXI.

5.1.4 Ontology Pros and Cons for Knowledge Representation with COMBATXXI

The ontology showed that it is possible to express knowledge semantically and have a reasoner classify a threat based on the clues (IED indicators) that a human would detect using a UAV such as the Raven. It is important to note that the cues used in the ontology can be represented in a model like COMBATXXI.

5.1.5 Future Work

The work presented in this thesis is an initial look at possible uses of combat models such as COMBATXXI. This work can be continued in many ways. Below are some possible areas that merit further investigation.

Develop extensions and support structures to allow COMBATXXI to be more amenable for use in studies that use the DOE methodology. This would allow it to be used in studies to look at complex parameter spaces that are not possible with the current structures.

Investigate the integration of ontology based KR systems into combat models such as COMBATXXI. By adding such structures, a wide range of investigation is made possible. These can range from identification of threats that take into consideration both observations and other information, such as past activity, to being able to reason about motives of opposing entities. The use of ontologies could also support more robust behaviors that can be used in a wider set of circumstances, which would facilitate building general purpose behaviors that can be reused in many scenarios.

APPENDIX A:
JOHNSON CRITERIA

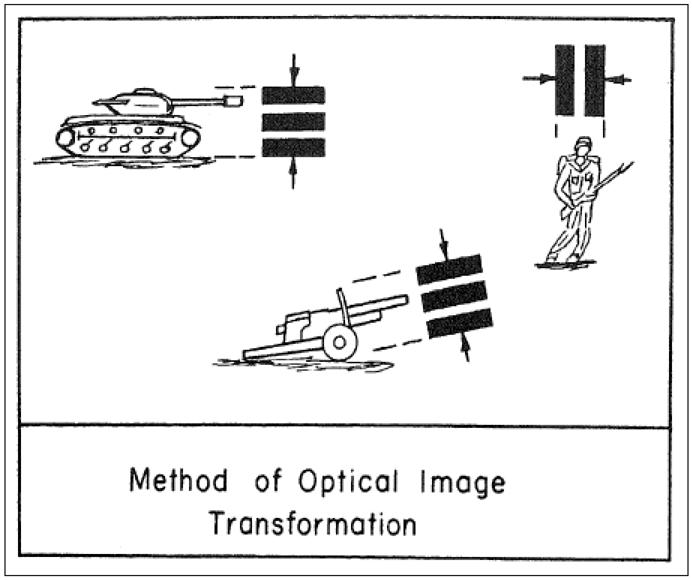

Figure A.1: Normalizing resolved line pairs for critical target dimension. After [44, p. 264]

TARGET	RESOLUTION PER MINIMUM DIMENSION			
Broadside View	Detection	Orientation	Recognition	Identification
Truck	.90	1.25	4.5	8.0
M-48 Tank	.75	1.2	3.5	7.0
Stalin Tank	.75	1.2	3.3	6.0
Centurion Tank	.75	1.2	3.5	6.0
Half-track	1.0	1.50	4.0	5.0
Jeep	1.2	1.50	4.5	5.5
Command Car	1.2	1.5	4.3	5.5
Soldier (Standing)	1.5	1.8	3.8	8.0
105 Howitzer	1.0	1.5	4.8	6.0
Average	1.0 ± .25	1.4 ± .35	4.0 ± .8	6.4 ± 1.5

Figure A.2: Number of minimum line pairs required for the four detection, orientation, recognition, identification. From [44, p. 264]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: CONCEPT OF OPERATIONS

Mission

In support of 2nd Platoon (PLT), Alpha Company (Co.), 2-37 IN (Infantry), 4th squad provides security and logistic runs in partnership with the 3rd LNA Battalion to support operations for the local populace while protecting the Dam.

Situation

2nd Platoon, A CO, 2-37 IN, conducted a relief in place (RIP) of the combat outpost (COP) three months ago. 2nd Platoon is scheduled to RIP with 1st Platoon within the next week and will assume FOB security at the CO HQ and serve as the CO reserve. Besides working and training the LNA troops, the platoon is also responsible for protecting the Dam workers that live on the COP. The missions are broken down into equal parts and each squad will rotate responsibilities to reduce complacency. The four task are (1) Entry Control Point [3 personnel x 3 eight hour. shifts = 1 squad], (2) Force Protection [3 Towers /OP points x 3 eight hour shifts = 1 squad], (3) Dam Observation Post [3 personnel x 3 eight hour shifts = 1 squad], and (4) local security patrols and quick reaction force = 1 squad. For each squad there is one LNA platoon supporting to be the immediate face to the community. 4th squad is equipped with 1 x Raven 11Q B UAS. Four of the 11 soldiers in the squad are qualified Raven operators.

Commander's Intent

2nd Platoon partnered with 3rd Battalion LNA will protect the population from insurgents, mentor the LNA partners, and protect the LNA partners in contact with insurgents. This partnership will maintain force protection at all times and provide over watch to the Dam to safeguard the area's power supply. 4th Squad will perform logistic runs in support of (ISO) 2nd Platoon, A CO, 2-37 IN.

Enemy

Threat forces consist of 300 insurgents equipped with RPG, PKM, AK-47, HMG, AT-3, SPG-9, SA-16 and 82mm mortar. The main threat consists of 3 x PLTs with two supporting efforts. Each PLT has the ability to emplace 2 x IEDs. There are reports of 8 to 10-man insurgent teams equipped with RPG, AK-47. There are also reports of possible IED emplacements along MSR WEST VIRGINIA.

B.0.6 Terrain

Open desert to desert mountains with adjacent urban structures.

B.0.7 Troops Available

Coalition and Host Nation Forces: 1 x U.S. Infantry squad equipped with organic direct fire weapons, with support from 120mm mortars. 21 soldiers from the LNA company and 16 from the LNP.

B.0.8 Time

0800, day, clear.

B.0.9 Civilian

Civilian populace is either controlled or intimidated by enemy operating within the area of operations (AO).

APPENDIX C: CONVOY MOVE IN FORMATION

```
================================
Name: ConvoyMoveInFormation
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
    formation=>java.lang.String
    route=>cxxi.model.objects.features.CMPolyline
    NAI_phaseLines=>java.util.ArrayList
    UAS_routes=>java.util.ArrayList
    falseNegativeRate=>[Ljava.lang.String;@7b449b1d
    falsePositiveRate=>[Ljava.lang.String;@5523cc24
Metadata:
Code:
 ================================
 Name: initInfo
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     if _gt_activeNode.getVar("isInited") == None:
       _gt_activeNode.putVar("isInited", 1)
       _htn_precon_ret=1
  ================================
  Name: isCommander
  AllowMsg: true
  Node Type: DEFAULT
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
```

```
  if state.isCommander():
   _htn_precon_ret=1
==============================
Name: setup
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
   from cxxi.model import CombatXXIModel
   from os import access
   from os import F_OK
   from os import W_OK
   from sys import stderr
   try:
    logfile
    writeOK
   except NameError:
    # open the log file in the same place CXXI puts the other log files
    logfilePathName = str(CombatXXIModel.getOutputDirectory()) + "/DFP.log"
    writeOK = True
    try:
     logfile = open(logfilePathName,"w")
    except IOError:
     print >>stderr,"\n\nWarning␣!!!!␣Could␣not␣open␣log␣file␣to␣write␣–␣log␣will␣not␣be␣output.\n\n"
     print >>stderr,"␣␣␣␣␣␣␣␣␣␣␣␣␣Log␣file␣may␣be␣open␣in␣another␣application.\n\n"
     writeOK = False
    if writeOK:
     _gt_activeNode.putVar("logfile", logfile)
   # get the rep number for this run
   repNum = CombatXXIModel.getReplicationNumber()
   _gt_activeNode.putVar("repNum", repNum)
   # set up the holder in the borg to hold the non–ied names
   name = "notIED"
   notIED = []
   borg.addAttribute(name, notIED, _gt_activeNode)
```

```
================================
Name: initCommander
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
   from HTN import UtilityFuncsExp
   #state.addControlMeasure(nai) #adds control measure for CXXI to pay attention to.
   for cm in _gt_activeNode.getParam("NAI_phaseLines"):
    state.addControlMeasure(cm.getName())
   speed = 2
   # scehdule the event to get us going
   UtilityFuncsExp.scheduleEvent(
      info.getMyAssignedName(),
      "GoalTracker_MoveOut",
      0.001,
      speed)
================================
Name: addReplanTriggers
AllowMsg: true
Node Type: INTERRUPT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
   goalContainer.getCurrentExecutingStack().addReplanTrigger("AtAControlMeasure")
   goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_MoveOut")
   goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_UASPickedUp")
   goalContainer.getCurrentExecutingStack().addReplanTrigger("EmbarkComplete")
   goalContainer.getCurrentExecutingStack().addReplanTrigger("DamageOccurredInMyUnit")
```

```
    goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_ResumeMovingAtSpeed")
    goalContainer.getCurrentExecutingStack().addReplanTrigger("StoppedMovement")
 ================================
 Name: doNothing
 AllowMsg: true
 Node Type: GOAL
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     '''
     At this point only the commander does anything so if the tree is assigned to someone who is
     not the commander, just end the tree.
     NOTE - if this tree is to be used if there is attrition this should not be
     done this way and there should be a way to keep the tree around in case the current
     "non-commander" is promoted to command via attrition.
     '''
================================
Name: events
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
 ================================
 Name: isGoalTrackerEvent
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
```

```
  if state.getLastTrigger().startswith("doGoalTracker_"):
    _htn_precon_ret=1
================================
Name: isMoveOut
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
   if state.getLastTrigger() == "doGoalTracker_MoveOut":
     _htn_precon_ret=1
     printMessage("GOAL TRACKER EVENT"+state.getLastTrigger(), True)
 ================================
 Name: moveOut
 AllowMsg: true
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     from cxxi.model.behavior import OrderUtilities
     from java.util import Vector
     from mtry.cxxi.model.HierarchicalTaskNetwork import HTNUtilities
     formation = _gt_activeNode.getParam("formation")
     route = _gt_activeNode.getParam("route")
     printMessage("formation is " + formation, True)
     params = state.getLastTriggerParams()
     speed = float(params[0])
     ord = HTNUtilities._py_getMoveOrderFromRoute(route, speed, formation)
     orders.addOrder(ord)
     startTime = state.getSimTime()
     _gt_activeNode.putVar("startTime", startTime)
================================
Name: isGoalTracker_UASPickedUp
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
```

```
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger() == "doGoalTracker_UASPickedUp":
     _htn_precon_ret=1
 ==============================
 Name: pickedUpUAS
 AllowMsg: true
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     from HTN import UtilityFuncsExp
     # we have picked up the UAS, so we should continue
     observations = state.getLastTriggerParams()[0]
     if observations.isEmpty():
      UtilityFuncsExp.scheduleEvent(
          info.getMyAssignedName(),
          "GoalTracker_ResumeMovingAtSpeed",
          0.001,
          None)
     else:
      # look at what the UAV sent us
      # this is where the processing of the knowledge could go.  If the info the UAS
      # sent back not clear other info could be used
      # for now we kill the IED and delay while it is destroyed
      whoToKill = observations.get(0).getAssignedName()
      #whoToKill= str((UtilityFuncs.getEntityById(IDofWhoToKill)).getAssignedName())
      state.killEntity(whoToKill,"CATASTROPHIC_KILL")
      UtilityFuncsExp.scheduleEvent(
          info.getMyAssignedName(),
          "GoalTracker_ResumeMovingAtSpeed",
          600.0,
          None)
==============================
Name: isResumeMovingAtSpeed
```

```
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    if state.getLastTrigger() == "doGoalTracker_ResumeMovingAtSpeed":
     _htn_precon_ret=1
  ================================
  Name: resumeMoving
  AllowMsg: true
  Node Type: INTERRUPT
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
     speedToResume = _gt_activeNode.getVar("speedToResume")
     UtilityFuncsExp.changeOrderSpeed(info.getMySelf(),speedToResume)
================================
Name: modelEvents
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
 ================================
 Name: isAtControlMeasure
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    if state.getLastTrigger()== "doAtAControlMeasure":
     _htn_precon_ret=1
  ================================
```

```
Name: isAtPhaseLine
AllowMsg: true
Node Type: INTERRUPT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    from cxxi.model.objects.holders import CMHolder
    from java.util import Vector
    from cxxi.model.behavior import OrderUtilities
    cm = state.getLastTriggerParams()[0]
    NAI_phaseLines = _gt_activeNode.getParam("NAI_phaseLines")
    for i in range(NAI_phaseLines.size()):
     aCMName = NAI_phaseLines.get(i).getName()
     if aCMName == cm.getName():
      printMessage("AT" + str(cm.getName()), True)
      uasRouteToUse = _gt_activeNode.getParam("UAS_routes")[i]
      _gt_activeNode.putVar("uasRouteToUse", uasRouteToUse)
      _htn_precon_ret=1
 ================================
 Name: stopAndWait
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     speedToResume = state.getCurrentSpeed()
     _gt_activeNode.putVar("speedToResume", speedToResume)
     UtilityFuncsExp.changeOrderSpeed(info.getMySelf(),0.0001)
 ================================
 Name: dispatchUAS
 AllowMsg: true
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
```

```
        from HTN import UtilityFuncsExp
        printMessage("Start UAS Tree", True)
        #airRoute = CMHolder.retrieveControlMeasureByName(
        "AIR_CORRIDOR_UAS_NAI_I_ROUTE")
        uasRouteToUse = _gt_activeNode.getVar("uasRouteToUse")
        # set the path to the goal
        goalPath = "HTN/Trees/OperateUAS.xml"
        UtilityFuncsExp.addGoal(
            "B0000c01A1",
            1.0,
            goalPath,
            [uasRouteToUse,
            _gt_activeNode.getParam("falsePositiveRate"),
            _gt_activeNode.getParam("falseNegativeRate")],
            None)
==============================
Name: isDamageOccured
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger() == "doDamageOccurredInMyUnit":
     _htn_precon_ret=1
==============================
Name: stopConvoy
AllowMsg: true
Node Type: GOAL
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    from cxxi.model.behavior import OrderUtilities
    from java.util import Vector
    #if damage has occured in this unit we need to stop and for this scenario we are done
```

```
    if state.isCommander():
        _htn_precon_ret=1
        prims=Vector()
        prims.add(OrderUtilities.createStopPrimitive())
        ord=orders.createOrder("HTN Plan Auto Generated", 0.0, prims)
        orders.preemptAllOrders(ord)
        simtime = state.getSimTime()
        elapsedTime = (simtime - _gt_activeNode.getVar("startTime"))/60.0
        printMessage("mission ended after " + str(elapsedTime) + " minutes", True)
        # log this run
        logfile = _gt_activeNode.getVar("logfile")
        fnr = _gt_activeNode.getParam("falseNegativeRate")
        fpr = _gt_activeNode.getParam("falsePositiveRate")
        repNum = _gt_activeNode.getVar("repNum")
        logfile.write('%d\t%.2f\t'%(repNum,simtime))
        logfile.write('Fail\t')
        logfile.write('%.2f\t%.2f\t%.2f\n'%(elapsedTime,fnr,fpr))
        logfile.close()
==============================
Name: isStoppedMoving
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger() == "doStoppedMovement":
        _htn_precon_ret=1
==============================
Name: doneWithMission
AllowMsg: true
Node Type: GOAL
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    from cxxi.model import CombatXXIModel
    simtime = state.getSimTime()
```

```
        elapsedTime = (simtime - _gt_activeNode.getVar("startTime"))/60.0
        printMessage("finished mission at " + str(elapsedTime), True)
        # log this run
        logfile = _gt_activeNode.getVar("logfile")
        fnr = _gt_activeNode.getParam("falseNegativeRate")
        fpr = _gt_activeNode.getParam("falsePositiveRate")
        repNum = _gt_activeNode.getVar("repNum")
        logfile.write('%d\t%.2f\t'%(repNum,simtime))
        logfile.write('Success\t')
        logfile.write('%.2f\t%.2f\t%.2f\n'%(elapsedTime,fnr,fpr))
        #CombatXXIModel.stopModel("run complete", 0)
        logfile.close()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: OPERATE THE IED HTN

```
==============================
Name: OperateIED
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
    sensor=>[Ljava.lang.String;@66ba60c7
    sensorRange=>[Ljava.lang.String;@5627dd81
    targetCount=>[Ljava.lang.String;@533f6c57
    P_K_KIll=>[Ljava.lang.String;@68e1ee73
Metadata:
Code:
    ''' Notes
    This is a tree to operate a simple IED.
    '''
  ==============================
  Name: initInfo
  AllowMsg: true
  Node Type: DEFAULT
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
      if _gt_activeNode.getVar("isInited") == None:
        _gt_activeNode.putVar("isInited", 1)
        _htn_precon_ret=1
    ==============================
    Name: isRealIED
    AllowMsg: true
    Node Type: DEFAULT
    Is Code File: false
    Import:
    Datamap:
    Metadata:
```

```
Code:
   if _gt_activeNode.getParam("P_K_KIll") > 0.0:
    _htn_precon_ret=1
 ==============================
 Name: startObservation
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    # set the the number of entities that have entered the AOI to zero
    _gt_activeNode.putVar("entitiesEnteredSensorAOI", 0)
    sensor = _gt_activeNode.getParam("sensor")
    sensorRange = _gt_activeNode.getParam("sensorRange")
    sensorObj = obs.getSensorByName(sensor)
    sensorObj.setEffectiveRange(sensorRange)
    printMessage("  Turning on area sensor!", True)
    if obs.startAOISensor(sensor):
     sensorObj = obs.getSensorByName(sensor)
     printMessage("  AOI sensor turned on", True)
    else:
     print info.getMyAssignedName(),":  Cannot start AOI Sensor "
 ==============================
 Name: addReplanTriggers
 AllowMsg: false
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    # model events
    goalContainer.getCurrentExecutingStack().addReplanTrigger("EnteredAOI")
    goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_DetonateNow")
==============================
Name: setIsNotIEDInfo
AllowMsg: true
```

```
  Node Type: GOAL
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
      '''
      Since this is not an actual IED - only something that could be confused as an
      an IED, it does not need any behavior so this tree can finish.
      '''
      borg.notIED.append(str(info.getMyAssignedName()))
      printMessage(" is not and IED", True)
================================
Name: events
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
  ================================
  Name: isGoalTrackerEvent
  AllowMsg: true
  Node Type: DEFAULT
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
      if state.getLastTrigger().startswith("doGoalTracker_"):
        _htn_precon_ret=1
    ================================
    Name: isDetonateNow
    AllowMsg: true
    Node Type: INTERRUPT
    Is Code File: false
    Import:
    Datamap:
    Metadata:
```

```
 Code:
    printMessage("GOAL␣TRACKER␣EVENT"+state.getLastTrigger(), True)
    if state.getLastTrigger() == "doGoalTracker_DetonateNow":
     _htn_precon_ret=1
  ===============================
  Name: detonate
  AllowMsg: true
  Node Type: GOAL
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
     from UtilityFuncs import getUniformRandomNumber
     # check to see if we need to kill an entity
     rn = getUniformRandomNumber(0,1.0)
     if rn <= _gt_activeNode.getParam("P_K_KIll"):
      # the most recent to enter the area is the entity that will be killed.
      IDofWhoToKill = state.getLastTriggerParams()[0]
      whoToKill= str((UtilityFuncs.getEntityById(IDofWhoToKill)).getAssignedName())
      state.killEntity(whoToKill,"CATASTROPHIC_KILL")
     # have the IED destroy itself
     printMessage("IED␣self␣destruct␣RND␣=␣" + str(rn), True)
     state.killEntity(str(info.getMyAssignedName()),"CATASTROPHIC_KILL")
===============================
Name: modelEvents
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
 ===============================
 Name: isEnteredAOI
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
```

```
Datamap:
Metadata:
Code:
   if state.getLastTrigger() == "doEnteredAOI" :
    _htn_precon_ret=1
 ==============================
 Name: entityEnteredAOI
 AllowMsg: false
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    from HTN import UtilityFuncsExp
    import UtilityFuncs
    entitiesEnteredSensorAOI = _gt_activeNode.getVar("
 entitiesEnteredSensorAOI")
    entitiesEnteredSensorAOI += 1
    if  entitiesEnteredSensorAOI >= _gt_activeNode.getParam("targetCount
 "):
     printMessage("Hello,␣I␣am␣executing␣the␣isEnteredAOI␣node␣and␣the␣
 sim␣time␣is:␣␣"     +str(state.getSimTime()), True)
     entityID = state.getLastTriggerParams()[1]
     UtilityFuncsExp.scheduleEvent(
         info.getMyAssignedName(),
         "GoalTracker_DetonateNow",
         0.001,
         entityID) # pass the id of the entity that my be killed
    else:
     printMessage("Not␣enough␣entities␣im␣my␣AOI␣yet", True)
     _gt_activeNode.putVar("entitiesEnteredSensorAOI",
 entitiesEnteredSensorAOI)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E:
OPERATE UAS HTN

```
================================
Name: OperateUAS
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
    route=>cxxi.model.objects.features.CMPolyline
    falsePositiveRate=>java.lang.Double
    falseNegativeRate=>java.lang.Double
Metadata:
Code:
 ================================
 Name: initInfo
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     if _gt_activeNode.getVar("isInited") == None:
       _gt_activeNode.putVar("isInited", 1)
       _htn_precon_ret=1
  ================================
  Name: addReplanTriggers
  AllowMsg: true
  Node Type: DEFAULT
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
      #goalContainer.getCurrentExecutingStack().addReplanTrigger("TakeoffDone")
```

```
    goalContainer.getCurrentExecutingStack().addReplanTrigger("HaveLanded")
    goalContainer.getCurrentExecutingStack().addReplanTrigger("EmbarkComplete")
    goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_CompletedFOVScan")
================================
Name: createUASRoutes
AllowMsg: true
Node Type: INTERRUPT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    from HTN import UtilityFuncsExp
    from cxxi.model.behavior import OrderUtilities
    from java.util import Vector
    from java.util import LinkedHashSet
    from mtry.cxxi.model.HierarchicalTaskNetwork import HTNUtilities
    from cxxi.model.physics.geometry import Direction
    from HTN.Scripts.general import SensorHelpers
    myTransporter = myTransport.getTransporter()
    _gt_activeNode.putVar("myTransporter", myTransporter)
    allObs = LinkedHashSet()
    _gt_activeNode.putVar("allObs", allObs)
    # get off the transporter so we can fly our mission
    myTransport.debark()
    route = _gt_activeNode.getParam("route")
    startPoint = state.getCurrentLocation()
    _gt_activeNode.putVar("startPoint", startPoint)
    prims = Vector()
    prims.addAll(HTNUtilities._py_createMovePOsFromRoutes(route, 10.0))
    prims.add(OrderUtilities.createMovePrimitive(startPoint, 10.0))
    prims.add(OrderUtilities.createStopPrimitive())
    ord = orders.createOrder("HTN Plan Auto Generated", 0.0, prims)
    orders.addOrder(ord)
    printMessage("Ready to turn on UAS sensor", True)
    UtilityFuncsExp.addGoal(
        info.getMyAssignedName(),
        2.0,
```

```
        "HTN/Trees/UAVSensor.xml",
        [0.0, -45.0,20.0,1000.0,3.5,1.0],
        None)
    outbound = True
    _gt_activeNode.putVar("outbound", outbound)
    lastDistToEndPoint = 0.0
    _gt_activeNode.putVar("lastDistToEndPoint", lastDistToEndPoint)
================================
Name: events
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
 ================================
 Name: isGoalTrackerEvent
 AllowMsg: true
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    if state.getLastTrigger().startswith("doGoalTracker_"):
     _htn_precon_ret=1
  ================================
  Name: isCompletedFOVScan
  AllowMsg: true
  Node Type: INTERRUPT
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
     if state.getLastTrigger() == "doGoalTracker_CompletedFOVScan":
      _htn_precon_ret=1
   ================================
   Name: checkSensor
   AllowMsg: true
```

```
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
    #
    # check to see if we are close to the start point in which case we do not continue scanning
    currentLoc = state.getCurrentLocation()
    dist = currentLoc.distanceTo(_gt_activeNode.getVar("startPoint"))
    outbound = _gt_activeNode.getVar("outbound")
    lastDistToEndPoint = _gt_activeNode.getVar("lastDistToEndPoint")
    # test if we are still outbound
    if outbound and dist < lastDistToEndPoint:
     # we have started back to the landing point
     outbound = False
     _gt_activeNode.putVar("outbound", outbound)
    else:
     lastDistToEndPoint = dist
     _gt_activeNode.putVar("lastDistToEndPoint", lastDistToEndPoint)
    if dist > 100.0 or outbound:
     UtilityFuncsExp.addGoal(
        info.getMyAssignedName(),
        0.0001,
        "HTN/Trees/UAVSensor.xml",
        [0.0,-45.0,30.0,1000.0,3.5,1.0],
        None)
    observations = state.getLastTriggerParams()[0]
    allObs = _gt_activeNode.getVar("allObs")
    allObs.addAll(observations)
================================
Name: modelEvents
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
 ================================
```

```
Name: isHaveLanded
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger() == "doHaveLanded":
      _htn_precon_ret=1
 ==============================
 Name: UASLanded
 AllowMsg: true
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
      from cxxi.model.behavior import OrderUtilities
      from java.util import Vector
      printMessage("I␣have␣landed", True)
      myTransporter = _gt_activeNode.getVar("myTransporter")
      prims=Vector()
      prims.add(
    OrderUtilities.createBoardEntityPrimitiveByName(
    myTransporter.getAssignedName()))
      ord=orders.createOrder("HTN␣Plan␣Auto␣Generated", 0.0, prims)
      orders.addOrder(ord)
==============================
Name: isHaveEmbarked
AllowMsg: true
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger() == "doEmbarkComplete":
      _htn_precon_ret=1
 ==============================
```

```
Name: doneWithMission
AllowMsg: false
Node Type: GOAL
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    from HTN import UtilityFuncsExp
    from java.util import ArrayList
    printMessage("embarked on transport", True)
    allObs = _gt_activeNode.getVar("allObs")
    obsList = ArrayList()
    # now process the observations and see if we have anything to report
    for something in allObs:
     aname = something.getAssignedName()
     rn = getUniformRandomNumber(0,1.0)
     printMessage("checking obs of " + str(aname) + " rnd num is " + str(rn), True)
     # check to see ground truth if this is an IED
     if borg.notIED.count(aname) == 0:
      # this is not in the non-IED list
      if rn > _gt_activeNode.getParam("falseNegativeRate"):
       # we have correctly identified this as an IED
    # so add it to the observations
       obsList.add(something)
     else:
      if rn < _gt_activeNode.getParam("falsePositiveRate"):
       # we think this is an IED so add it to the observations
       obsList.add(something)
    # tell our transported that we have been loaded
    UtilityFuncsExp.scheduleEvent(
        _gt_activeNode.getVar("myTransporter").getAssignedName(),
        "GoalTracker_UASPickedUp",
        0.001,
        obsList)
```

APPENDIX F:
UAV SENSING HTN

```
================================
Name: UAVSensor
AllowMsg: false
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
    sensorAzimuth=>[Ljava.lang.String;@2831300d
    sensorPitch=>[Ljava.lang.String;@3549ba18
    FOVWidth=>[Ljava.lang.String;@7b5898fc
    maxRange=>[Ljava.lang.String;@75a407a7
    meanFOVTime=>[Ljava.lang.String;@63f2a147
    standDevFOVTime=>[Ljava.lang.String;@4e3a6f94
Metadata:
Code:
    '''␣Notes
␣␣␣This␣is␣a␣simple␣model␣for␣a␣sensor␣observing␣a␣single␣Field␣of␣View.
␣␣␣'''
 ================================
 Name: initInfo
 AllowMsg: false
 Node Type: DEFAULT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     if _gt_activeNode.getVar("isInited") == None:
       _gt_activeNode.putVar("isInited", 1)
       _htn_precon_ret=1
  ================================
  Name: initsearch
  AllowMsg: false
  Node Type: DEFAULT
  Is Code File: false
  Import:
```

```
Datamap:
Metadata:
Code:
    from HTN import UtilityFuncsExp
    from UtilityFuncs import getNormalRandomNumber
    #
    # compute a time to scan this FOR
    #
    scanTime = getNormalRandomNumber(_gt_activeNode.getParam("meanFOVTime"),
      _gt_activeNode.getParam("standDevFOVTime"))
    if scanTime < 1.0:
      scanTime = 0.0
    _gt_activeNode.putVar("scanTime", scanTime)
    printMessage("scan time is " + str(scanTime), True)
    UtilityFuncsExp.scheduleEvent(
        info.getMyAssignedName(),
        "GoalTracker_StartFOVScan",
        0.001,
        None)
===============================
Name: addReplanTriggers
AllowMsg: false
Node Type: INTERRUPT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_StartFOVScan")
    goalContainer.getCurrentExecutingStack().addReplanTrigger("GoalTracker_FOVScanDone")
===============================
Name: events
AllowMsg: false
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
```

```
Code:
================================
Name: isGoalTrackerEvent
AllowMsg: false
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger().startswith("doGoalTracker_"):
        _htn_precon_ret=1
================================
Name: isStartFOVScan
AllowMsg: false
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    if state.getLastTrigger() == "doGoalTracker_StartFOVScan":
        _htn_precon_ret=1
================================
Name: scanFOV
AllowMsg: false
Node Type: INTERRUPT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
    from HTN import UtilityFuncsExp
    from HTN.Scripts.general.SensorHelpers import isInFOV
    from UtilityFuncs import getUniformRandomNumber
    # collect what we can see in this FOV
    from java.util import ArrayList
    observations = ArrayList()
    _gt_activeNode.putVar("observations", observations)
    maxRange = _gt_activeNode.getParam("maxRange")
    thingsCloseBy =myIntel.getLiveEntitiesInRadius(maxRange * 1.1)
```

```
        myLocation = state.getCurrentLocation()
        # get the actual sensor orientation
        sensorOrientation = state.getCurrentHeading()
        sensorOrientation.addDegreesToHeading(_gt_activeNode.getParam("sensorAzimuth"))
        sensorOrientation.setPitchInDegrees(_gt_activeNode.getParam("sensorPitch"))
        for something in thingsCloseBy:
          tgtLoc = something.getPhysicalState().getSynchedLocation(False)
          side = something.getSide()
          if str(side) == "BLUE":
            continue
          if isInFOV(tgtLoc, myLocation, sensorOrientation,
            _gt_activeNode.getParam("FOVWidth"), maxRange):
            # compute probability of seeing
            distToTgt = myLocation.distanceTo(tgtLoc)
            pObs = 1.0 - (distToTgt/(220))
            rn = getUniformRandomNumber(0,1.0)
            printMessage("In FOV " + str(something.getAssignedName()) +
          "  Prob  detect is " + str(pObs), True)
            if rn < pObs:
              observations.add(something)
              printMessage("I saw " + str(something.getAssignedName()) +
            "  rn is " + str(rn), True)
        UtilityFuncsExp.scheduleEvent(
              info.getMyAssignedName(),
              "GoalTracker_FOVScanDone",
              _gt_activeNode.getVar("scanTime"),
              None)
================================
Name: isFOVScanDone
AllowMsg: false
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
      if state.getLastTrigger() == "doGoalTracker_FOVScanDone":
        _htn_precon_ret=1
 ================================
```

```
  Name: scanDone
  AllowMsg: false
  Node Type: GOAL
  Is Code File: false
  Import:
  Datamap:
  Metadata:
  Code:
      from HTN import UtilityFuncsExp
      UtilityFuncsExp.scheduleEvent(
          info.getMyAssignedName(),
          "GoalTracker_CompletedFOVScan",
          0.001,
          _gt_activeNode.getVar("observations"))
================================
Name: modelEvents
AllowMsg: false
Node Type: DEFAULT
Is Code File: false
Import:
Datamap:
Metadata:
Code:
 ================================
 Name: printEvent
 AllowMsg: false
 Node Type: INTERRUPT
 Is Code File: false
 Import:
 Datamap:
 Metadata:
 Code:
     printMessage("EVENT"+state.getLastTrigger(), True)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G:
IED ONTOLOGY


```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.ied.com/ontologies/ied.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.ied.com/ontologies/ied.owl">
    <Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <Literal datatypeIRI="&rdf;PlainLiteral">An IED ontology that
    describes various IEDs based on different properties that make up an IED
    .</Literal>
    </Annotation>
    <Declaration>
        <Class IRI="#Abandoned"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Batteries"/>
    </Declaration>
    <Declaration>
```

```
            <Class IRI="#BelowGround"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Cable"/>
        </Declaration>
        <Declaration>
            <Class IRI="#CellPhone"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Child"/>
        </Declaration>
        <Declaration>
            <Class IRI="#ChildSBIED"/>
        </Declaration>
        <Declaration>
            <Class IRI="#ConfirmedIED"/>
        </Declaration>
        <Declaration>
            <Class IRI="#DeliveryMethod"/>
        </Declaration>
        <Declaration>
            <Class IRI="#DetectionMeans"/>
        </Declaration>
        <Declaration>
            <Class IRI="#DisturbedSoil"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Electric"/>
        </Declaration>
        <Declaration>
            <Class IRI="#ElectricCurrent"/>
        </Declaration>
        <Declaration>
            <Class IRI="#FemaleSBIED"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Grenade"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Human"/>
        </Declaration>
```

```
<Declaration>
    <Class IRI="#IED"/>
</Declaration>
<Declaration>
    <Class IRI="#IEDIndicators"/>
</Declaration>
<Declaration>
    <Class IRI="#IsolatedBox"/>
</Declaration>
<Declaration>
    <Class IRI="#Location"/>
</Declaration>
<Declaration>
    <Class IRI="#Magnetic"/>
</Declaration>
<Declaration>
    <Class IRI="#MaleSBIED"/>
</Declaration>
<Declaration>
    <Class IRI="#Man"/>
</Declaration>
<Declaration>
    <Class IRI="#ManWithCellPhone"/>
</Declaration>
<Declaration>
    <Class IRI="#MetalPlate"/>
</Declaration>
<Declaration>
    <Class IRI="#MiltaryVehicle"/>
</Declaration>
<Declaration>
    <Class IRI="#MouseTrap"/>
</Declaration>
<Declaration>
    <Class IRI="#Munitions"/>
</Declaration>
<Declaration>
    <Class IRI="#NonMilitaryVehicle"/>
</Declaration>
<Declaration>
    <Class IRI="#NonShellMilitary"/>
```

```
    </Declaration>
    <Declaration>
        <Class IRI="#Occupied"/>
    </Declaration>
    <Declaration>
        <Class IRI="#OnGround"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Package"/>
    </Declaration>
    <Declaration>
        <Class IRI="#PackageIED"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Physical"/>
    </Declaration>
    <Declaration>
        <Class IRI="#PossibleIED"/>
    </Declaration>
    <Declaration>
        <Class IRI="#PressurePlate"/>
    </Declaration>
    <Declaration>
        <Class IRI="#RoundRPG"/>
    </Declaration>
    <Declaration>
        <Class IRI="#RoundSmallArms"/>
    </Declaration>
    <Declaration>
        <Class IRI="#SBIED"/>
    </Declaration>
    <Declaration>
        <Class IRI="#ShellArtillery"/>
    </Declaration>
    <Declaration>
        <Class IRI="#ShellMilitary"/>
    </Declaration>
    <Declaration>
        <Class IRI="#ShellMortar"/>
    </Declaration>
    <Declaration>
```

```
        <Class IRI="#SodaCan"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Springs"/>
    </Declaration>
    <Declaration>
        <Class IRI="#SteelTubes"/>
    </Declaration>
    <Declaration>
        <Class IRI="#StrongBeliefIED"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Touch"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Trigger"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Trip"/>
    </Declaration>
    <Declaration>
        <Class IRI="#VBIED"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Vehicle"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Vehicles"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Visual"/>
    </Declaration>
    <Declaration>
        <Class IRI="#WeakBeliefIED"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Wires"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Woman"/>
    </Declaration>
```

```
<Declaration>
    <ObjectProperty IRI="#hasDeliveryMethod"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasIEDIndicators"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasLocation"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasPackageDeliveryMethod"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasPossibleBelief"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasTrigger"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasVehicleDeliveryMethod"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasIEDIndicatorName"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#isConfirmed"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#disturbedsoil_1"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#ied_0"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#ied_1"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#ied_2"/>
```

```
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#ied_3"/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#isolatedbox_1"/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#wires_1"/>
    </Declaration>
    <EquivalentClasses>
        <Class IRI="#ConfirmedIED"/>
        <ObjectIntersectionOf>
            <Class IRI="#IED"/>
            <DataHasValue>
                <DataProperty IRI="#isConfirmed"/>
                <Literal datatypeIRI="&xsd;boolean">true</Literal>
            </DataHasValue>
        </ObjectIntersectionOf>
    </EquivalentClasses>
    <EquivalentClasses>
        <Class IRI="#IED"/>
        <ObjectIntersectionOf>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="#hasDeliveryMethod"/>
                <Class IRI="#DeliveryMethod"/>
            </ObjectSomeValuesFrom>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="#hasIEDIndicators"/>
                <Class IRI="#IEDIndicators"/>
            </ObjectSomeValuesFrom>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="#hasLocation"/>
                <Class IRI="#Location"/>
            </ObjectSomeValuesFrom>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="#hasTrigger"/>
                <Class IRI="#Trigger"/>
            </ObjectSomeValuesFrom>
        </ObjectIntersectionOf>
    </EquivalentClasses>
```

```
<EquivalentClasses>
    <Class IRI="#IEDIndicators"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasIEDIndicatorName"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#PossibleIED"/>
    <ObjectIntersectionOf>
        <Class IRI="#IED"/>
        <DataHasValue>
            <DataProperty IRI="#isConfirmed"/>
            <Literal datatypeIRI="&xsd;boolean">false</Literal>
        </DataHasValue>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#StrongBeliefIED"/>
    <ObjectIntersectionOf>
        <Class IRI="#PossibleIED"/>
        <ObjectMinCardinality cardinality="3">
            <ObjectProperty IRI="#hasIEDIndicators"/>
            <Class IRI="#IEDIndicators"/>
        </ObjectMinCardinality>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#WeakBeliefIED"/>
    <ObjectIntersectionOf>
        <Class IRI="#PossibleIED"/>
        <ObjectMinCardinality cardinality="1">
            <ObjectProperty IRI="#hasIEDIndicators"/>
            <Class IRI="#IEDIndicators"/>
        </ObjectMinCardinality>
        <ObjectMaxCardinality cardinality="2">
            <ObjectProperty IRI="#hasIEDIndicators"/>
            <Class IRI="#IEDIndicators"/>
        </ObjectMaxCardinality>
    </ObjectIntersectionOf>
</EquivalentClasses>
```

```
<SubClassOf>
    <Class IRI="#Abandoned"/>
    <Class IRI="#Vehicles"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Batteries"/>
    <Class IRI="#IEDIndicators"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#BelowGround"/>
    <Class IRI="#Location"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Cable"/>
    <Class IRI="#Wires"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CellPhone"/>
    <Class IRI="#Trigger"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Child"/>
    <Class IRI="#Human"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#ChildSBIED"/>
    <Class IRI="#SBIED"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#ChildSBIED"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
        <Class IRI="#Child"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#DisturbedSoil"/>
    <Class IRI="#IEDIndicators"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Electric"/>
```

```
        <Class IRI="#Wires"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#ElectricCurrent"/>
        <Class IRI="#Trigger"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#FemaleSBIED"/>
        <Class IRI="#SBIED"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#FemaleSBIED"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
            <Class IRI="#Woman"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Grenade"/>
        <Class IRI="#NonShellMilitary"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Human"/>
        <Class IRI="#DeliveryMethod"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#IsolatedBox"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Magnetic"/>
        <Class IRI="#DetectionMeans"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#MaleSBIED"/>
        <Class IRI="#SBIED"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#MaleSBIED"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
```

```
            <Class IRI="#Man"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Man"/>
        <Class IRI="#Human"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#ManWithCellPhone"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#MetalPlate"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#MiltaryVehicle"/>
        <Class IRI="#Vehicle"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#MouseTrap"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Munitions"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NonMilitaryVehicle"/>
        <Class IRI="#Vehicle"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NonShellMilitary"/>
        <Class IRI="#Munitions"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Occupied"/>
        <Class IRI="#Vehicles"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#OnGround"/>
```

```
        <Class IRI="#Location"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Package"/>
        <Class IRI="#DeliveryMethod"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PackageIED"/>
        <Class IRI="#PossibleIED"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PackageIED"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasPackageDeliveryMethod"/>
            <Class IRI="#Package"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Physical"/>
        <Class IRI="#DetectionMeans"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PressurePlate"/>
        <Class IRI="#Trigger"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#RoundRPG"/>
        <Class IRI="#NonShellMilitary"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#RoundSmallArms"/>
        <Class IRI="#NonShellMilitary"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#SBIED"/>
        <Class IRI="#IED"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#SBIED"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
```

```
                <Class IRI="#Human"/>
            </ObjectSomeValuesFrom>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#ShellArtillery"/>
            <Class IRI="#ShellMilitary"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#ShellMilitary"/>
            <Class IRI="#Munitions"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#ShellMortar"/>
            <Class IRI="#ShellMilitary"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#SodaCan"/>
            <Class IRI="#IEDIndicators"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#Springs"/>
            <Class IRI="#IEDIndicators"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#SteelTubes"/>
            <Class IRI="#IEDIndicators"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#Touch"/>
            <Class IRI="#Trigger"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#Trip"/>
            <Class IRI="#Wires"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#VBIED"/>
            <Class IRI="#IED"/>
        </SubClassOf>
        <SubClassOf>
            <Class IRI="#VBIED"/>
```

```
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasVehicleDeliveryMethod"/>
            <Class IRI="#Vehicle"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Vehicle"/>
        <Class IRI="#DeliveryMethod"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Vehicles"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Visual"/>
        <Class IRI="#DetectionMeans"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Wires"/>
        <Class IRI="#IEDIndicators"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Woman"/>
        <Class IRI="#Human"/>
    </SubClassOf>
    <DisjointClasses>
        <Class IRI="#Abandoned"/>
        <Class IRI="#Occupied"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#Batteries"/>
        <Class IRI="#DisturbedSoil"/>
        <Class IRI="#IsolatedBox"/>
        <Class IRI="#ManWithCellPhone"/>
        <Class IRI="#MetalPlate"/>
        <Class IRI="#MouseTrap"/>
        <Class IRI="#Munitions"/>
        <Class IRI="#SodaCan"/>
        <Class IRI="#Springs"/>
        <Class IRI="#SteelTubes"/>
        <Class IRI="#Vehicles"/>
```

```
        <Class IRI="#Wires"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#BelowGround"/>
        <Class IRI="#OnGround"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#Cable"/>
        <Class IRI="#Electric"/>
        <Class IRI="#Trip"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#CellPhone"/>
        <Class IRI="#ElectricCurrent"/>
        <Class IRI="#PressurePlate"/>
        <Class IRI="#Touch"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#Child"/>
        <Class IRI="#Man"/>
        <Class IRI="#Woman"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#ChildSBIED"/>
        <Class IRI="#FemaleSBIED"/>
        <Class IRI="#MaleSBIED"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#ConfirmedIED"/>
        <Class IRI="#PossibleIED"/>
        <Class IRI="#SBIED"/>
        <Class IRI="#VBIED"/>
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#DeliveryMethod"/>
        <Class IRI="#DetectionMeans"/>
        <Class IRI="#IED"/>
        <Class IRI="#IEDIndicators"/>
        <Class IRI="#Location"/>
        <Class IRI="#Trigger"/>
    </DisjointClasses>
```

```
<DisjointClasses>
    <Class IRI="#Grenade"/>
    <Class IRI="#RoundRPG"/>
    <Class IRI="#RoundSmallArms"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Human"/>
    <Class IRI="#Package"/>
    <Class IRI="#Vehicle"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Magnetic"/>
    <Class IRI="#Physical"/>
    <Class IRI="#Visual"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#MiltaryVehicle"/>
    <Class IRI="#NonMilitaryVehicle"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#NonShellMilitary"/>
    <Class IRI="#ShellMilitary"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#PackageIED"/>
    <Class IRI="#SBIED"/>
    <Class IRI="#VBIED"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#ShellArtillery"/>
    <Class IRI="#ShellMortar"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#StrongBeliefIED"/>
    <Class IRI="#WeakBeliefIED"/>
</DisjointClasses>
<ClassAssertion>
    <Class IRI="#DisturbedSoil"/>
    <NamedIndividual IRI="#disturbedsoil_1"/>
</ClassAssertion>
<ClassAssertion>
```

```
        <Class IRI="#IED"/>
        <NamedIndividual IRI="#ied_0"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#IED"/>
        <NamedIndividual IRI="#ied_1"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#IED"/>
        <NamedIndividual IRI="#ied_2"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#IED"/>
        <NamedIndividual IRI="#ied_3"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#IsolatedBox"/>
        <NamedIndividual IRI="#isolatedbox_1"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Wires"/>
        <NamedIndividual IRI="#wires_1"/>
    </ClassAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_1"/>
        <NamedIndividual IRI="#disturbedsoil_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_1"/>
        <NamedIndividual IRI="#wires_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_2"/>
        <NamedIndividual IRI="#isolatedbox_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_2"/>
```

```
        <NamedIndividual IRI="#wires_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_2"/>
        <NamedIndividual IRI="#disturbedsoil_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_3"/>
        <NamedIndividual IRI="#wires_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_3"/>
        <NamedIndividual IRI="#isolatedbox_1"/>
    </ObjectPropertyAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <NamedIndividual IRI="#ied_3"/>
        <NamedIndividual IRI="#disturbedsoil_1"/>
    </ObjectPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasIEDIndicatorName"/>
        <NamedIndividual IRI="#disturbedsoil_1"/>
        <Literal datatypeIRI="&xsd;string">dirt</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#isConfirmed"/>
        <NamedIndividual IRI="#ied_0"/>
        <Literal datatypeIRI="&xsd;boolean">false</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#isConfirmed"/>
        <NamedIndividual IRI="#ied_1"/>
        <Literal datatypeIRI="&xsd;boolean">false</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#isConfirmed"/>
        <NamedIndividual IRI="#ied_2"/>
        <Literal datatypeIRI="&xsd;boolean">false</Literal>
```

```
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#isConfirmed"/>
        <NamedIndividual IRI="#ied_3"/>
        <Literal datatypeIRI="&xsd;boolean">true</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasIEDIndicatorName"/>
        <NamedIndividual IRI="#isolatedbox_1"/>
        <Literal datatypeIRI="&xsd;string">isolatedbox</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasIEDIndicatorName"/>
        <NamedIndividual IRI="#wires_1"/>
        <Literal datatypeIRI="&xsd;string">wires</Literal>
    </DataPropertyAssertion>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
        <ObjectProperty IRI="#hasDeliveryMethod"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#hasPackageDeliveryMethod"/>
        <ObjectProperty IRI="#hasDeliveryMethod"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#hasVehicleDeliveryMethod"/>
        <ObjectProperty IRI="#hasDeliveryMethod"/>
    </SubObjectPropertyOf>
    <FunctionalObjectProperty>
        <ObjectProperty IRI="#hasLocation"/>
    </FunctionalObjectProperty>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasDeliveryMethod"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
```

```
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasIEDIndicators"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasLocation"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasPackageDeliveryMethod"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasPossibleBelief"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasTrigger"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasVehicleDeliveryMethod"/>
        <Class IRI="#IED"/>
    </ObjectPropertyDomain>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasDeliveryMethod"/>
        <Class IRI="#DeliveryMethod"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasHumanDeliveryMethod"/>
        <Class IRI="#Human"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasLocation"/>
        <Class IRI="#Location"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasPackageDeliveryMethod"/>
        <Class IRI="#Package"/>
```

```
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasPossibleBelief"/>
        <Class IRI="#PossibleIED"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasTrigger"/>
        <Class IRI="#Trigger"/>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasVehicleDeliveryMethod"/>
        <Class IRI="#Vehicle"/>
    </ObjectPropertyRange>
    <FunctionalDataProperty>
        <DataProperty IRI="#hasIEDIndicatorName"/>
    </FunctionalDataProperty>
    <FunctionalDataProperty>
        <DataProperty IRI="#isConfirmed"/>
    </FunctionalDataProperty>
    <DataPropertyDomain>
        <DataProperty IRI="#hasIEDIndicatorName"/>
        <Class IRI="#IEDIndicators"/>
    </DataPropertyDomain>
    <DataPropertyDomain>
        <DataProperty IRI="#isConfirmed"/>
        <Class IRI="#IED"/>
    </DataPropertyDomain>
    <DataPropertyRange>
        <DataProperty IRI="#hasIEDIndicatorName"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataPropertyRange>
    <DataPropertyRange>
        <DataProperty IRI="#isConfirmed"/>
        <Datatype abbreviatedIRI="xsd:boolean"/>
    </DataPropertyRange>
</Ontology>



<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net
    -->
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] U. S. Army, *COMBATXXI Users Guide*, U.S. Army WSMR, n.d.

[2] T. Mauer et al., “Search and detection modeling of military imaging systems,” in *Proc. SPIE 5784, Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XVI, 201*, Orlando, Florida, 2005, pp. 201–215.

[3] I. Balogh et al., “Using hierarchical task networks to create dynamic behaviors in combat models,” unpublished.

[4] S. C. for Biomedical Informatics Research. *What Is Protege?* The National Center for Biomedical Ontology. (2013). [Online]. Available: http://protege.stanford.edu/aboutus/aboutus.html

[5] U. Staff, *Eyes of the Army, U.S. Army Unmanned Aircraft Systems Roadmap 2010-2035*, U.S. Army UAS Center of Excellence, Fort Rucker, Alabama, n.d.

[6] J. Gertler, “U.S. unmanned aerial systems,” Congressional Research Service, CSR Report for Congress R42136, Jan. 2012.

[7] Raven overview, (2013). [Online]. Available: http://www.avinc.com/downloads/Raven_Gimbal.pdf

[8] Raven UAS launch, (2013). [Online]. Available: http://www.avinc.com/img/media_gallery/UAS_Raven_Field_Launch.jpg

[9] Kestrel - land mti for small unmanned aircraft systems, (2013). [Online]. Available: http://www.avinc.com/uas/kestrel/

[10] U. A. Force, “U.S. Air Force scan eagle fact sheet,” (2013). [Online]. Available: http://www.af.mil/AboutUs/FactSheets/Display/tabid/224/Article/104532/scan-eagle.aspx

[11] AAI unmanned aircraft systems. Shadow RQ-7B System Data Sheet. Hunt Valley, MD, 2012.

[12] Shadow 200 rq-7 tactical unmanned aircraft system, United States of America. [Online]. Available: http://www.army-technology.com/projects/shadow200uav/shadow200uav3.html

[13] General atomics aeronautical. Gray Eagle Armed Persistence Data Sheet. Poway: CA, G.A.A.S.I., 2012.

[14] General atomics awarded $30m gray eagle contract. (2012). [Online]. Available: http://www.unmanned.co.uk/unmanned-vehicles-news/unmanned-aerial-vehicles-uav-news/general-atomics-awarded-30m-gray-eagle-contract/

[15] U.S. Air Force mq-9 reaper fact sheet. (2010). [Online]. Available: http://www.af.mil/AboutUs/FactSheets/Display/tabid/224/Article/104470/mq-9-reaper.aspx

[16] MQ-9 reaper. photo: Air force, (2012). [Online]. Available: http://www.wired.com/dangerroom/2012/04/killer-drone-upgrade/training-to-hunt/

[17] U.S.J.C. of Staff, *Joint Publication 3-07.2 Antiterrorism*, Department of Defense, Washington, D.C.

[18] Faces of the fallen. (2013). [Online]. Available: http://apps.washingtonpost.com/national/fallen/causes-of-death/ied/

[19] U. A. Training and Doctrine, *FM 3-24.2 Tactics in Counterinsurgency*, Headquarters Department of the Army, Washington, D.C.

[20] C. Shepherd, "Methods for ied reconnaissance and detection," vol. 113, p. not cited, Sep./Oct. 2004.

[21] R. H. Vollmerhausen and E. Jacobs, "The targeting task performance (ttp) metric a new model for predicting target acquisition performance," Modeling and Simulation Division Night Vision and Electronic Sensors Directorate, VA, Tech. Rep. AMSEL-NV-TR-230, Aug. 2004.

[22] O. H. Schade, "Optical and photoelectric analog of the eye," *J. Opt. Soc. Am.*, vol. 46, no. 9, pp. 721–738, Sep 1956. [Online]. Available: http://www.opticsinfobase.org/abstract.cfm?URI=josa-46-9-721

[23] Night vision and electronic sensors directorate history. (2013). [Online]. Available: http://www.nvl.army.mil/history.html

[24] Thermal imaging: how far can you see it? data sheet. (2013). [Online]. Available: http://www.flir.com/uploadedfiles/eng_01_howfar.pdf

[25] B. O'Kane et al., "Cycle criteria for detection of camouflaged targets," DTIC Document, Tech. Rep., 2005.

[26] C. J. Darken, "Computer graphics-based models of target detection: Algorithms, comparison to human performance, and failure modes," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 4, no. 3, pp. 262–276, 2007. [Online]. Available: http://dms.sagepub.com/content/4/3/262.abstract

[27] F. Baez, "Target acquisition methods in combat simulations," unpublished.

[28] E. D. Sacerdoti, "A structure for plans and behavior," DTIC Document, Tech. Rep., 1975.

[29] S. S. J. A. Baier and S. A. McIlraith, "Htn planning with preferences."

[30] I. Balogh et al., “Use of hierarchical task networks to model complex operational tasks in combatxxi(PowerPoint Presentation),” 80th Military Operations Research Society Symposium, Mission Area Analysis Branch, 3300 Russell Road Quantico, VA, June 11-14, 2012.

[31] R. Brachman and H. Levesque. (2003.) "Knowledge representation and reasoning". [Online]. Available: http://www.cin.ufpe.br/~mtcfa/files/in1122/Knowledge%20Representation%20and%20Reasoning.pdf

[32] R. Davis et al.,“What Is a Knowledge Representation,” *AI Magazine*, vol. 14, no. 1, pp. 17–33, 1993.

[33] V. Mehta, D. Patel, and N. Reddy. History of knowledge representation. (2008). [Online]. Available: http://www.knowledgerep.info/history.html

[34] K. Klement. Propositional logic: History. (2005). [Online]. Available: "http://www.iep.utm.edu/prop-log/#H2"

[35] G. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Boston, MA: Addison-Wesley, 2009.

[36] A. Collins and M. Quillian, “Retrieval time from semantic memory,” *Journal of Verbal Learning and Verbal Behavior*, vol. 8, no. 2, pp. 240–247, 1969.

[37] Merriam-Webster. Merriam-webster. (2013). [Online]. Available: http://www.merriam-webster.com/dictionary/ontology

[38] C. Childers, “Applying semantic web concepts to support net-centric warfare using the tactical assessment markup language TAML,” M. S. thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, 2006.

[39] TRAC-Monterey, “Deployable force protection scenario,” unpublished.

[40] I. Balogh, “UAV sensor model,” PowerPoint Slide, Naval Postgraduate School, Monterey, CA, Aug 2013.

[41] W.W.W. Consortium. Owl web ontology language overview, introduction. (2004). [Online]. Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/

[42] Semantic reasoner. (2013). [Online]. Available: "http://en.wikipedia.org/wiki/Semantic_reasoner"

[43] M. Horridge. *A practical guide to building owl ontologies using protege 4 and co-ode tools edition 1.3. (2011).* [Online]. Available: "http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf"

[44] J. Johnson, "Analysis of image forming systems," presented at the Image Intensifier Symposium, Warfare Vision Branch, Electrical Engineering Dept., U.S. Army Engineer Research and Development Laboratories, 1958.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia
2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California