GMTI Modelling and Encoding

James McAvoy
Senior ISR Systems Engineer (Contractor)
DLCSPM 4-PMO ISTAR C2
4th Floor, Louis St-Laurent Building
555 Boul. de la Carrière
Gatineau, Quebéc K1A 0K2
Telephone: 613-836-9209
Mobile: 613-220-7041
Email: jimcavoy@thetastream.com

Abstract—STANAG 4607 defines Ground Moving Target Indicator (GMTI) message format as an ordered sequence of physical bits and bytes, where the positions of the syntactical elements are fixed. Defining a message in this manner has caused issues with the radar sensor community. Deployed systems based on older editions of STANAG 4607 would fail to decode messages from later editions because the sequence of bits and bytes will be considered out of order by the receiver. To solve this issue, industry has employed Model Driven Engineering (MDE) using ASN.1 for last twenty years to describe their communication protocols with notable success. The question being proposed is; can Model Driven Engineering approach improve GMTI system develop life-cycle? We conducted an experiment with limited objectives to find out. The experimental results were encouraging. This study is part of an overall effort to investigate alternative approaches of defining a GMTI format for both specification and encoding.

Index Terms—GMTI, ASN.1, STANAG 4607, Model Driven Engineering (MDE)

1 Introduction

SENSOR radars transmit moving target detections formatted as *Ground Moving Target Indicator* (GMTI). To ensure interoperability between NATO members, NATO publishes STANAG 4607 [1], *NATO GMTI Format*. The standard describes the binary layout for detection data and also provides a format to request surveillance services.

STANAG 4607 is a message format where it outlines a well-defined set of messages each of which carries a defined meaning (semantics), together with the rules for exchanging these messages and processing these messages between end systems. The standard does not specify a transmission method, but the AEDP 7 does provide guidance for transmitting over UDP either broadcast or multicast. Usually, sensors transmit GMTI messages over a TCP/IP protocol stack using UDP at the Transport Layer but can be provided as files as done with the sensors on a Heron. In summary, STANAG 4607 specifies a message format for moving target detections that is designed to be independent of transmission method.

- *J. McAvoy is working under contract for the Department of National Defence, Canada.*
  E-mail: jimcavoy@thetastream.com

Manuscript received 11 April 2012. This work was supported by Director Land Command Systems Program Management 4 (DLCSPM 4) under contract PSA.AGR.1536.1752. This paper was presented in part of STANAG 4545 CST meeting, April 2012.

1.1 The Problem

Presently, we believe there are two issues in the manner how the sensor community defines the GMTI format:

- the specification is an unstructured document that prevents automation; and
- the encoding format is inflexible to change.

The following subsection will cover these issues in more detail.

1.1.1 *Specification*

Presently, the GMTI format specification is an unstructured (i.e. not machine processable) text document expressing the GMTI message structure using human readable statements. Although the specification is well written, human developers have to read, interpret, and manually code the specification, which can be labor intensive, and error prone activity. The result, the deployed systems may exhibit unpredictable behaviour when interworking with other deployed systems created from different implementers.

If the GMTI format specification was declared in a structured format (i.e. machine processable and computable), programs will be responsible to generate code instead of human developers. This could improve overall system predictability, interoperability and cost savings.

1.1.2 *Encoding*

STANAG 4607 is a binary-based format where the message structure is defined as string of octets or bits.

© 2012 Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence. All rights reserved.

Each syntactical message element is represented as octets in the GMTI format that has a fixed position, order and length. New features added to the standard will introduce a new syntactical element into the GMTI format. This could cause end systems to decode the GMTI message incorrectly because the end system expects the syntactical elements to be in a prescribe order, position and length in the bitstream. This manner of encoding GMTI messages is causing migration and extensibility issues. For example, older deploy systems will fail to decode GMTI messages based on a newer edition of the standard.

We could propose the GMTI format standard be specified as a character-based protocol, where the message structure is defined as a series of lines of ASCII encoded text. Each syntactical element in the message is represented as a line of ASCII text. When a new feature is added, the decoder can ignore the new syntactical elements in the message it fails to understand. Two examples of very successful character-based protocol are HTTP [9] and SMTP [10]. The Internet relays heavily on the HTTP protocol, while Email on SMTP. These protocols are more tolerate to migration and extensibility issues but consume more bandwidth then binary-based protocols. Most radar systems operate in a very constraint and limit bandwidth network environment, which makes character-based protocols unsuitable.

We desire a GMTI format to be a binary-based because of its small footprint on bandwidth. At the same time, the format should exhibit character-based protocol behaviour for migration and extensibility advantages.

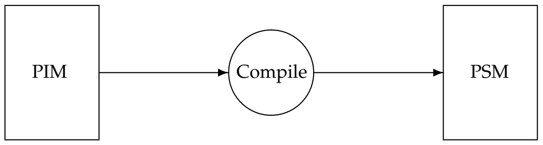
1.2 Motivation

If one surveys standards that various standard bodies publish today, one will discover the protocols and message formats are expressed using some type of notation or modelling language such as UML [15], XML Schema [12], ABNF [8] or ASN.1 [4]. The modelling language defines the *abstract syntax* of the message that is independent of implementation. This decoupling from protocol specification and encoding implementation provides the advantage to protocol designers to produce specification without undue concern with the encoding issues.

Since the specification is expressed using a machine processable language (i.e. formal language), application independent tools can be utilized to generate platform dependent code for message serialization. Because the model is independent to the tools that generate the serialization code (i.e. bits on the wire), it facilitates migration to improved encoding methods when they become available without changing the specification.

The other positive side of effect of using a modelling language for protocol development, it reduces ambiguity and increases clarity of the standard. Since machines poorly handle ambiguous statements, it imposes protocol designers to be more rigorous during specification, which could lead to better standards. Protocol designers can use applications to automate the process of discovering errors in the specification before the standard is published. Hopefully, reducing the time and cost of reviewing standards.

Fig. 1. Model Driven Engineering (MDE) approach using models for software development.



In the Motion Imagery domain, video systems transmit metadata using Key-Length-Value (KLV) triplets, which prove to be very flexible and easy method of encoding. SMPTE 336M-2003 [7] defines the use of KLV in Motion Imagery. KLV borrows heavily from BER (Basic Encoding Rule) that is part of in ASN.1 standard. We reasoned that we should investigate ASN.1 to specify a GMTI format and take advantage of *Model Driven Engineering* (MDE) approach to standards and system development.

1.2.1 Model Driven Engineering

Model Drive Engineering is a software development methodology which focuses on creating and processing models to enhance clarity of specifications, to promote communication between individuals and teams, and to automate the production of code during implementation. Using MDE, standards specifiers create a *Platform Independent Model* (PIM) using modelling language or domain specific language (DSL) like ASN.1 or UML. Then developers compile the PIM to generate *Platform Specific Model* (PSM), which can be data structures written in C or Java programming language for a target operating system (i.e. Linux, Windows) and hardware platform (i.e. Intel x86, ARM).

By using this engineering approach, we hope to increase predictability and reduce cost during specification and system implementation. Machine processable models and programs will create the target systems instead humans hand-crafting code from informal and unstructured text documents, which is error prone and costly. In appendix C, we conducted a cost analysis of code generated from an ASN.1 compiler when reading a ASN.1 specification of a GMTI format based on STANAG 4607 that demonstrates the cost savings using this approach.

1.3 Goal

The goal of this study is to determine if Model Driven Engineering can improve GMTI system development life-cycle, from message specification to implementation and eventually sensor deployment. We will design and develop a simple GMTI system based on STANAG 4607 that uses ASN.1 standard for specification and encoding for our study.

1.4 Objectives

To determine if MDE approach can improve GMTI system development, we used the following four limited objectives:

- **Extensibility**. Can we extend the standard by adding new features into the specification without any negative side-effects to older implementations? We plan to design the GMTI format specification so that new features can be included into a new editions of the standard and still permit newer implementations to interwork with already deployed older implementations. For example, end system is based on edition A of the protocol and receives an edition B message.
- **Interoperability**. Stakeholders should have the freedom to select tools, programming language and computing platform to implement a GMTI system based on Platform Independent Model (i.e. format specification). In this study, we used ASN.1 C compilers from two different vendors to create the PSM from the GMTI PIM. We wanted to observe when an encoder create from one ASN.1 compiler can interwork with a decoder create from a different vendor.
- **Completeness and Correctness**. We planned to base the GMTI PIM on an existing STANAG 4607 standard. The study implemented an encoder that reads GMTI messages formatted in accordance with STANAG 4607 and outputs BER encoded GMTI messages. We hypothesized that the BER encoding of the GMTI message is isomorphic to STANAG 4607 encoded message.
- **Performance**. In this study, performance is related to how bandwidth efficient is ASN.1 BER encoding in comparison with previous GMTI encoding format. We hypothesized that BER encoding of a GMTI message will be at least twice as large the present GMTI format, but still more efficient than text based ASN.1 encoding methods.

1.5 Outline

The paper will first introduce a short summary of ASN.1. The section will cover the salient features of ASN.1 notation, concepts and technology. We will next discuss the GMTI model that we developed using ASN.1 notation. The section will cover how we mapped STANAG 4607 specification to create GMTI model based on ASN.1 notation. The experiment section covers our approach to determine if ASN.1 can meet our study's objectives. The results section outlines our observation based on the study's objectives. We summarizes our findings in the conclusion and provide recommendations for future studies.

2 A Short Overview of ASN.1

In telecommunication and computer networking domain, **Abstract Syntax Notation One** (ASN.1) [4] is standard that provides a flexible notation to describe your data structure for encoding, decoding and transmitting of messages over a network. It is a character based protocol that provides a set of formal rules for describing the structure of objects that are independent of machine-specific encoding techniques. The ASN.1 language is a precise and formal notation that ensures clarity and reduces ambiguities.

ASN.1 is a joint ISO/IEC and ITU-T standard that was originally defined 1984 as part of CCITT X.409:1984. It was later removed to its own standard and is covered in X.680 series. The latest available edition is dated 2002 and is backward compatible with 1995 edition.

ASN.1 notation defines the *abstract syntax* of the message that is independent of the way the information is encoded. In MDE context, ASN.1 notation specifies the PIM. Below example shows the abstract syntax or PIM for a Packet Header specified in ASN.1.

```
PacketHeader ::= SEQUENCE
 {
   version             OCTET STRING (SIZE(2)),
   nationality         IA5String,
   security            PacketSecurity,
   exerciseIndicator   OCTET STRING (SIZE(1)),
   platformID          IA5String,
   missionID           Uint32,
   jobID               Uint32,
   ...
 }
```

Various ASN.1 encoding rules provide the *transfer syntax* of the data values whose abstract syntax is described in ASN.1. Below shows example of a transfer syntax for a Packet Header in BER format.

```
30 2F A0 2B 80 02 32 30
81 02 43 41 A2 0C 80 01
06 81 02 43 41 82 03 00
20 00 83 01 00 84 0A 44
49 53 43 55 53 20 31 20
20 85 01 00 86 01 01 A1
00
```

The standard ASN.1 provides several encoding rules which are as follows:

- Basic Encoding Rules (BER)
- Canonical Encoding Rules (CER)
- Distinguished Encoding Rules (DER)
- XML Encoding Rules (XER)
- Packed Encoding Rules (PER)
- Generic String Encoding Rules (GSER)

In this study, we were concerned with BER format but we quickly surveyed the other encoding rules and provided a comparison in the result section.

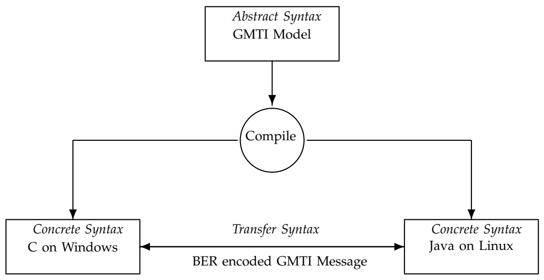
ASN.1 with its specific ASN.1 encoding rules facilitates the exchange of structured data especially between applications over networks by describing data structures in a manner that is independent of machine architecture and implementation language. Developers would compile an ASN.1 model to generate implementation language code for serialization that is specific to the machine architecture. The generated code defines the *concrete syntax* of the data values whose abstract syntax is described in a model defined by ASN.1. In other words, the developer generates a PSM from the GMTI PIM. Figure 2 shows the relationship between the three discussed ASN.1 syntax concepts. Below displays the concrete syntax of a Packet Header in the C programming language that an ASN.1 C compiler generated.

```
/*
 * Generated by asn1c-0.9.21 (http://lionet.info/asn1c)
 * From ASN.1 module "GMTIF"
 *  found in "stanag4607-ed3.asn"
 */

#ifndef _PacketHeader_H_
#define _PacketHeader_H_


#include <asn_application.h>

/* Including external dependencies */
#include <OCTET_STRING.h>
#include <IA5String.h>
#include "PacketSecurity.h"
#include "Uint32.h"
#include <constr_SEQUENCE.h>

#ifdef __cplusplus
extern "C" {
#endif

/* PacketHeader */
typedef struct PacketHeader {
    OCTET_STRING_t   version;
    IA5String_t  nationality;
    PacketSecurity_t     security;
    OCTET_STRING_t   exerciseIndicator;
    IA5String_t  platformID;
    Uint32_t     missionID;
    Uint32_t     jobID;
    /*
     * This type is extensible,
     * possible extensions are below.
     */

    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _asn_ctx;
} PacketHeader_t;

/* Implementation */
extern asn_TYPE_descriptor_t asn_DEF_PacketHeader;

#ifdef __cplusplus
}
#endif

#endif  /* _PacketHeader_H_ */
```

Several standard bodies specify their protocols on ASN.1. Following are some standards using ASN.1 for their specification:

- IETF RFC 3280 - Internet X.509 Public Key Infrastructure
- IETF RFC 3525 - Media Gateway Control
- IETF RFC 4511 - Lightweight Directory Access Protocol
- IETF RFC 3411 - Simple Network Management Protocol (SNMP)
- ITU-T X.400 - Electronic Mail
- ITU-T H.323 - Packet-Based Multimedia Communication Systems

Fig. 2. A figure showing the relationship between the three ASN.1 syntax concepts.



The Internet Engineering Task Force (IETF) specified some of their RFC using ASN.1 with human readable definitions associated with the model. There exist applications that extract ASN.1 model from the RFC. Later developers feed the extract model into an ASN.1 compiler to generate code for encoding and decoding.

2.1 Basic Encoding Rules (BER)

This sub-section provides a short overview of BER transfer syntax, which ASN.1 describes in ITU-T Rec. X.690 (BER, CER and DER) [5] standard. We mostly experimented with BER transfer syntax in this study.

The BER transfer syntax always has the format of a triplet **TLV**, often referred as Tag, Length, Value as in Figure 3(a). The Motion Imagery community barrows heavily on BER for metadata serialization, which they call **KLV** encoding. However, Motion Imagery community disregarded the abstract syntax component of ASN.1. Developers who are familiar with KLV encoding will understand BER encoding easily.

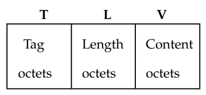
All the fields in **T**, **L** and **V** are series of octets. The value **V** can be a triplet **TLV** creating a recursive tree data structure, where interior nodes contain constructed values and the leaf nodes hold values of basic build-in types such as INTEGER or STRING (see Figure 3(b)). The transfer syntax is octet-based (i.e. binary based) and self-delimited since the field **L** provides a means of determining the length of each **TLV** triplet.

The **T**(ag) octets correspond to the encoding of the triplet's value type. One of the bits specifies the form of the **V** content octet as either *basic* or *constructed*. The tag octet is unique in its parent context. Generally, one octet is enough for a **T** field.

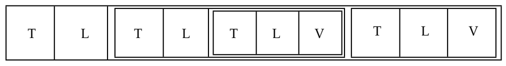
2.2 Alternative Approaches

Character-based notation provide an alternate approach to specify a communication protocol. Some of these character-based protocols, such as HTTP or SMTP, define messages using text tags and values based on Augmented Backus-Naur Form (ABNF) [8] notation. The definition also defines the encoding, which is in text. XML standards is another form of a character-based

Fig. 3. BER transfer syntax (**TLV** format)



(a) Triplet **TLV**



(b) Recursive principle

protocol. ASN.1 provides an XML encoding rule called XER that attempts to bridge the gap between binary to textual encoding. ITU-T Rec. X.694 [6] also provides rules to map XSD Schema to ASN.1 in order to exploit ASN.1 encoding rules. Therefore, models based on XSD Schema or UML can leverage ASN.1 encoding rules.

Character-based protocols offer transparency advantage over binary-base protocols. One only needs a text editor to see the content of a character-based message. However, character-based protocols consume more bandwidth than binary-based protocols. The result section will compare character and binary base encoding.

We plan to conduct future studies in other transfer syntax protocols such as *Fast Infoset* (FI) [13] and *Efficient XML Interchange* (EXI) [14], and compare them with ASN.1 encoding rules. These protocols specify how to binary encode XML documents.

We hypothesized that binary encode XML GMTI messages will impact runtime serialization process negatively. The reason is that the encoder has to create an XML document in memory or on the hard drive, which consumes CPU and IO resources. Then the XML document has to be serializes as binary encoded XML document before transmission, which consumes more CPU and IO resources. In contrast to ASN.1 encoders that serialize a GMTI message immediately using one the ASN.1 binary-based encoding rules, which utilize less computing resources. We plan to study this in the future to determine if this hypothesis is true.

3 GMTI Model

For this study, we created a specification of a GMTI format using ASN.1 (See appendix A). The GMTI format specification is based on STANAG 4607 edition 3. The mapping from the STANAG 4607 to our ASN.1 version of GMTI format was fairly easy and straight forward. Table 1 outlines how we mapped objects and types defined in STANAG 4607 to ASN.1 notation.

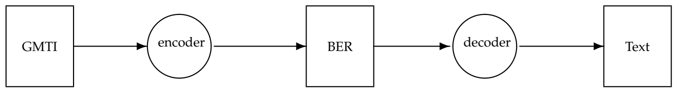
We only described a subset what is defined in STANAG 4607. The GMTI specification in appendix A, we specified *Packet Header*, *Mission Segment*, *Dwell Segment*, *Job Definition*, and *Test and Status Segment*. For each segment, we fully defined their STANAG 4607 definition.

In the model module definition, we enabled *automatic tagging* and *extensibility implied* features. By enabling

TABLE 1
This table shows the mapping for objects and types defined in STANAG 4607 to ASN.1 notation.

STANAG 4607	ASN.1
GMTI Message	SEQUENCE
Packet Header	SEQUENCE
Segment	CHOICE
Mission Segment	SEQUENCE
Dwell Segment	SEQUENCE
Job Definition Segment	SEQUENCE
Test and Status Segment	SEQUENCE
Alphanumeric	IA5String or OCTET STRING depending on the field
Integer	INTEGER with subtyping to constrain valid values
Binary Angles	REAL
Flags	BIT STRING
Enumeration	ENUMERATION

Fig. 4. The diagram showing the basic experimental setup where the rectangles represent files and the circles represent applications.



these features, the model designer is relieved from burden of explicitly assigning tags and deciding which objects can be extended.

Length fields defined in STANAG 4607 are unmapped because they are not required. The BER encoding rules provided this automatically for all syntactical elements in an GMTI message. We used the ASN.1 OPTIONAL keyword to denote which elements are optional in a GMTI message. Therefore, existence mask, like the one defined in a Dwell Segment, is absented in the ASN.1 GMTI model.

4 Experimental Approach

This section covers the experimental approach we employed for our study. The study's objectives were dealing mostly with ASN.1 notation and transfer syntax. Therefore, how well ASN.1 transfer syntax will operate over TCP/IP protocol stack and in a distributed network environment will be answered in future studies.

The basic experimental framework is fairly simple and can be replicated easily. We created several encoder and decoder command line interface (i.e. console) applications that accept files using the console input and produce their results onto console output. With this approach, interprocess communication between applications is achieved by file passing or chaining using pipes. Figure 4 shows the basic experimental framework.

TABLE 2
Sample files s_n that were used in the study.

Samples s_n	Size (Bytes)	Remarks
s_1	249	A GMTI message containing a Mission, Job Definition and Dwell segment with six Target Reports from a GMTI simulator.
s_2	223	A GMTI message containing a Job Definition and Dwell segment with two Target Reports from a GMTI simulator. See appendix B to see its content.
s_3	203,493	A GMTI message containing Dwell segment with 6776 Target Reports from a French Horizon sensor.
s_4	42,800	A GMTI message containing 24 Dwell segments and 5166 Target Reports from a live Heron sensor.
s_5	81,414	A GMTI message containing 50 Dwell segments and 9,814 Target Reports from a live Heron sensor.
s_6	2,698	A GMTI message containing 6 Dwell segments and 277 Target Reports from a live Heron sensor.
s_7	198,370	A GMTI message containing 3,929 Dwell segment and 380 Target Reports from a live PST sensor.
s_8	443	A GMTI message containing a Mission, Job Definition and six Dwell segment from a live PST sensor.
s_9	1,744,886	A file containing 1,421 GMTI messages from a UK sensor.
s_{10}	1,135,957	A file containing 931 GMTI messages from a NATO sensor simulator.
s_{11}	1,085,382	A file containing 900 GMTI messages from a NATO sensor simulator.
s_{12}	5,283,195	A file containing 14,168 GMTI messages from a live PST sensor, which was extracted from a Wireshark file that captured nearly an hour of network traffic.

4.1 Samples

In the study, we used an assortment of GMTI samples from various systems and nations. The GMTI samples were stored in binary files. Table 2 outlines the sample's file identifier, size and content.

4.2 ASN.1 Compilers

In this study, we experimented with ASN.1 C compilers that were developed by two different vendors. This section will provide a high-level overview of the two compilers we used in this study. There exists a variety of vendors that supply ASN.1 compiler for various programming languages, which we hope to research in the future.

4.2.1 Open Source ASN.1 Compiler

This is a freely available open source compiler that generates C source code from ASN.1 specifications. A single individual named Lev Walkin [17] developed this compiler. We like this compiler because it is free and source code is available for easy debugging and inspection. The code is portable across multiple platforms but for Windows, the compiler and generated code must be build under Cygwin [16], a Linux development environment for Windows. We successfully created an encoder and decoder application fairly easily, which we will name $e_{oss}(x)$ and $d_{oss}(x)$ respectively for this paper. Table 3 shows a short summary of this compiler's capabilities.

TABLE 3
Open source ASN.1 compiler's capabilities

Feature	Capabilities
Encoding	BER (CER,DER), XER, PER (unaligned only)
Target Language	C programming language
License	BSD
Cost	free

4.2.2 ASN Lab Compiler

ASN Lab [18] sells a commercial ASN.1 compiler. To use their compiler you need Eclipse and download their development tools (ADT), which is a collection of Eclipse plugins. In Eclipse, you create an ASN.1 project containing ASN.1 specification. The Eclipse ASN.1 plugins provides several convenient editing features, such as coding folding, outline views, text hovering, rename refactoring and others. We recommend their development tools for writing a ASN.1 specification because these editing features. Once you completed your ASN.1 specification, you invoke Eclipse build feature which automatically translates your specification into C code that can be integrated into your application.

To build (i.e. link) our applications with the generated C code, we needed to download their ASN.1 C runtime library, which they sell but provide a free trial version for limited time period. However, ASN Lab does not provide the source code for their C runtime library, which makes debugging and inspection impossible.

ASN Lab sells their ASN.1 C compiler product, which includes source code, documentation, technical support and unlimited updates for $12,000 USD.

In our study, we were able to create an encoder application but fail to create a decoder application. Poor documentation and lack of source code prevented us from creating our decoder application. We assumed if we had a full licence version of their product, we would be able to create a decoder application. In this paper, we will refer to this encoder as $e_{asnlab}(x)$.

During our study, we discovered a bug in how they BER encode REAL values. We reported the issue, which ASN Lab promptly fixed the bug and provided a new library the next day.

Table 4 shows a short summary of this compiler's capabilities.

4.3 Source Code and Sample Files

The source code for encoder and decoder applications, and sample files are available upon request.

TABLE 4
ASN Lab ASN.1 compiler's capabilities

Feature	Capabilities
Encoding	BER (CER,DER), PER (align and un-aligned)
Target Language	C programming language
License	Commercial
Cost	$12,000 USD

5 Results

This section outlines our experimental observations based on the study's objectives.

5.1 Extensibility

Can we have deployed systems based on different editions of a GMTI format interwork together without issues? We want decoders to read older versions of the GMTI message, and also be able to ignore newer introduced elements when reading later versions of a GMTI message.

In ASN.1 notation, protocol designers can define which components in the specification are extensible. To indicate that a type is extensible, one inserts an extension maker "..." in its definition. In ASN.1, the only extensible types are ENUMERATED, SEQUENCE, SET and CHOICE.

For this experiment, we extended the ASN.1 GMTI model outline in appendix A as follows:

1) Insert new basic component, *foobar*, of type INTEGER into the Packet Header definition, which is of type SEQUENCE.
2) Insert new constructed component, *ExtSegment*, which contains two elements of type INTEGER and IA5String. This will be a new GMTI Segment type that will extent the Segment definition of type CHOICE.

When decoding messages containing extensible type values, there are two cases:

- **Case A.** Unexpected extensions are present.
- **Case B.** The expected extensions are absent.

The following subsection covers the two cases that a decoder can encounter.

5.1.1 Case A

This experiment tries to prove that ASN.1 decoders can process GMTI message based on a newer edition of a specification. We built a new encoder, $e'_{oss}(x)$, based on a specification mention above. In the new encoder, we hard coded these new components, *foobar* and *ExtSegment*, to be insert in every GMTI message. Using the samples outline in table 2 and the new encoder, we created new BER encoded GMTI files containing the new components.

The decoder, $d_{oss}(x)$ represents an older deployed system based on previous edition of the specification. We feed $d_{oss}(x)$ with newly extended GMTI files that $e'_{oss}(x)$ produced. The decoder successfully transform the input file without issues. Our recorded observation is outlined in table 5.

TABLE 5
Recorded observation for the extensibility experiment. First column s_n are sample files outlined in table 2. The encoder inputs s_n and outputs s'_n, which decoder reads. The last column informs if the decoder can process the file s'_n successfully.

Case A			
s_n	**Encoder**	**Decoder**	**Process?**
s_1	$e'_{oss}(s_1)$	$d_{oss}(s'_1)$	Yes
s_2	$e'_{oss}(s_2)$	$d_{oss}(s'_2)$	Yes
s_8	$e'_{oss}(s_8)$	$d_{oss}(s'_8)$	Yes
Case B			
s_n	**Encoder**	**Decoder**	**Process?**
s_1	$e_{oss}(s_1)$	$d'_{oss}(s'_1)$	Yes
s_2	$e_{oss}(s_2)$	$d'_{oss}(s'_2)$	Yes
s_8	$e_{oss}(s_8)$	$d'_{oss}(s'_8)$	Yes

5.1.2 Case B

This experiment tries to prove that a newer decoder can process GMTI messages based on an older edition of a specification. We build a new decoder, $d'_{oss}(x)$, based on the new specification. Using encoder, $e_{oss}(x)$, that is based on older specification, we generated BER formatted GMTI message file. Then we inputted these files into the new decoder, $d'_{oss}(x)$, and recorded our observation.

We observed that the new decoder successfully process a GMTI message based on an older specification. Our recorded observation is outlined in table 5.

5.2 Interoperability

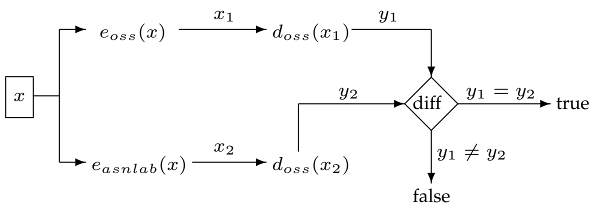
Can GMTI systems that were implemented using different ASN.1 compilers work together? This experiment attempts to answer this question.

As stated previously in the paper, we created two encoders, $e_{oss}(x)$ and $e_{asnlab}(x)$, from two different vendors and one decoder, $d_{oss}(x)$. To proof encoders are interoperable, the decoder, $d_{oss}(x)$, should be able to transform the encoders' output files exactly the same.

Let x be a file containing GMTI message encoded in accordance with STANAG 4607. In our limited experiment, $x \in S$ where $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ defined in table 2.

We inputted x into both encoders, $e_{oss}(x)$ and $e_{asnlab}(x)$, that outputted files containing BER encoded GMTI message x_1 and x_2 respectively. The decoder, $d_{oss}(x)$, transform these files to a GSER encoded files, y_1 and y_2. See appendix B for an example GSER encoded GMTI message. If $y_1 = y_2$ then encoders are

Fig. 5. The basic layout for the interoperability experiment. If decoder outputs $y_1 = y_2$, then encoders $e_{oss}(x)$ and $e_{asnlab}(x)$ are interoperable.



interoperable. Figure 5 displays the basic layout of this experiment. We used UNIX based **diff** utility to perform the comparison between y_1 and y_2.

We observed that the encoders $e_{oss}(x)$ and $e_{asnlab}(x)$ outputted well-formed and valid BER encoded GMTI files for all $x \in S$ but $x_1 \neq x_2$. In other words, both encoders generated different BER encoded bitstreams from the same GMTI file.

However, when x_1 and x_2 were transformed by decoder $d_{oss}(x)$ to produce y_1 and y_2, we observed $y_1 = y_2$ for all $x \in S$. Therefore, in this experiment, encoder $e_{oss}(x)$ and $e_{asnlab}(x)$ are interoperable. More study is required to determine this is universally true for all ASN.1 compilers.

5.3 Completeness and Correctness

For this experiment, we manually verified by comparing the content of the source GMTI file against encoder outputted BER or GSER files. From the sample set, we were successful to map 100% all GMTI message syntax elements to BER or GSER. We inspected all the values of each syntax element that the encoder produced against the source sample and observed that the values were equivalent. See appendix B for an example encoder output of a GMTI message.

During the experiment, we observed some source samples files outline in table 2 did not conform to the standard. This caused the encoders to produce semantically incorrect BER encoded GMTI message. To generate well-formed GMTI messages, we modified the encoders' parser to handle these exceptions. This was somewhat expected because developers must hand-craft code to implement encoders and decoders based on STANAG 4607, which can be error prone. We hope employing a MDE approach would reduce hand-crafted code and future systems will be based on machine generated encoders and decoders, which could increase system predictability and reliability.

We plan in future studies to develop a BER decoder application to populate a Coalition Share Database (CSD). We then can compare the BER encoded message against GMTI encoded message in the CSD and see if they are equivalent or equal.

TABLE 6
Table shows the file size in bytes and ratio of various encoding rules that the encoder $e_{oss}(x)$ generated and compares them with the files outlined in table 2.

s_n	BER	Ratio	GSER	Ratio	XER	Ratio
s_1	453	1.82	4,143	16.64	5,885	23.64
s_2	395	1.77	3,417	15.32	4,883	21.89
s_3	441,542	2.17	6,177,699	30.36	8,421,840	41.39
s_4	120,146	2.81	1,476,816	34.51	1,992,175	46.55
s_5	228,357	2.80	2,807,263	34.49	3,786,992	46.52
s_6	7,151	2.65	84,877	31.46	114,976	42.62
s_7	316,680	1.60	2,897,255	14.61	4,091,770	20.63
s_8	723	1.63	6,201	13.99	8,818	19.91
s_9	3,580,379	2.05	49,300,496	28.25	66,716,021	38.24
s_{10}	2,326,475	2.05	32,063,316	28.22	43,393,403	38.20
s_{11}	2,227,298	2.05	30,611,738	28.2	41,428,292	38.17
s_{12}	8,437,245	1.60	74,654,061	14.13	105,354,262	19.94
	Average	2.08		24.18		33.14

5.4 Performance

In this study, performance is related to the size of the files produced by the encoder applications. Table 6 and 7 shows the size of each file that encoder applications, $e_{oss}(x)$ and $e_{asnlab}(x)$, produced. We feed these encoders with files containing GMTI formatted message(s) that are outlined in table 2.

We hypothesized that BER encoding to be less efficient than STANAG 4607 encoding in size, which was true. We were surprised that the BER encoded GMTI messages was not much larger than the original in some cases. Generally, both encoders produced a GMTI message bitstream that were approximately 1.98 times larger than the original. Encoder $e_{asnlab}(x)$ produced slightly smaller files than $e_{oss}(x)$ when serializing a GMTI message using BER encoding. See table 6 and 7 to compare.

One would assume the that two encoders would produce files of equal size when using the same BER encoding method, but that was not the case. We discovered that the two encoders BER serialize values of type ASN.1 REAL differently, which is an explanation for the size differences. Both encoders generated files that conformed to ASN.1 BER standards.

As we expected, the character-based encoding rules produced larger file size. GSER encoded messages were 23.92 times larger than the original GMTI message while XER encoded messages were 32.89 times larger.

BER encoding may not be as efficient as GMTIF encoding but we gain improved predictability, interoperability and extensibility. We believe these advantages out weight the GMTIF size advantage. When comparing BER encoding with other text-based encoding, there is an substantial size advantage. We plan to study PER encoding, which claims to be more size efficient than BER, in future studies.

We also wanted to know how efficient the code that is generated from these ASN.1 compilers. Using sample GMTI file s_{12} described in table 2, we feed this file into the encoders and recorded the processing time. The

TABLE 7
Table compares the file size in bytes and ratio of BER encoding that the encoder $e_{asnlab}(x)$ generated and compares them with the files defined in table 2.

s_n	GMTIF	BER	Ratio
s_1	249	433	1.74
s_2	223	377	1.69
s_3	203,493	429,624	2.11
s_4	42,800	95,441	2.23
s_5	81,414	181,498	2.23
s_6	2,698	5778	2.14
s_7	198,370	295,751	1.49
s_8	443	687	1.55
s_9	1,744,886	3,479,807	1.99
s_{10}	1,135,957	2,261,355	1.99
s_{11}	1,085,382	2,165,201	1.99
s_{12}	5,283,195	7,977,169	1.51
		Average	1.89

TABLE 8
Table outlines the processing time of the two encoders when generating BER encoded GMTI messages.

		$e_n(x)$ Processing Time	
Files s_n	Message Count	$e_{oss}(x)$ (seconds)	$e_{asnlab}(x)$ (seconds)
s_9	1421	5	2
s_{10}	931	3	1
s_{11}	900	3	1
s_{12}	14,168	6	3

sample file s_{12} contains 14,168 GMTI messages that was from a Wireshark file capturing nearly an hour (3,582 seconds) of network traffic. Both encoders process this file extremely fast. The encoder, $e_{oss}(x)$, took six (6) seconds to complete, while encoder, $e_{asnlab}(x)$ took three (3) seconds. We can safely imply that ASN.1 based encoders will perform within time constraint requirements that are necessary for proper GMTI system behaviour. This assumption will be need to be confirm in future studies. Table 8 outlines our recorded observations on processing times when the two encoders read GMTI samples containing more than one message.

6 Conclusion

This paper presented a study to determine if Model Driven Engineering can improve GMTI system development life-cycle. We designed and developed a simple GMTI system using ASN.1 for specification and encoding for our study. To explore this approach, we conduct various experiments based on four objectives: 1) extensibility, 2) interoperability, 3) completeness and correctness, and 4) performance.

We first expressed a GMTI format specification based on STANAG 4607 using ASN.1 notation, which defines the abstract syntax of the GMTI message. ASN.1 provides a notation to describe the message format that is independent of encoding technique. From the specification, we were able to implement encoder and decoder applications from generated code that ASN.1 compiler produced, which proved save time and development effort.

With these encoder and decoder applications, we conducted experiments based on the objectives. Our experimental observations were very encouraging.

We discovered that encoders and decoders based on generated ASN.1 code supported extensibility. Encoders and decoders were able to interwork together with GMTI messages of different editions. We will investigate alternative approaches, such as binary encoded XML in future studies.

We used two different ASN.1 compiler vendors to determine interoperability. Again, encoders and decoders were able to interwork together with GMTI messages from different implementations. We plan to test other ASN.1 compiler vendors and implementation languages, such Java, in future studies.

We found that ASN.1 notation is rich enough (complete) to describe a GMTI specification based on STANAG 4607. We manually validated and verified (correctness) that the BER encoded GMTI messages was equivalent to a GMTI message that is formatted in accordance with STANAG 4607. In future studies, we plan to automate this process by transmitting BER encoded GMTI messages to a CSD.

We compared various ASN.1 encoding techniques. We observed that BER encoding created a GMTI bitstream that was approximately 1.9 times larger than STANAG 4607 encoded bitstreams. ASN.1 provides other binary-based encoding rules that are more size efficient than BER that we would like to investigate in future studies.

We quickly profiled the encoders processing speed. The encoders were able to process a 14,168 GMTI messages that represented nearly an hour of network traffic in less than 6 seconds. We are confident that encoders based on generated ASN.1 compiler code will be able to meet time constraints that are imposed by most GMITI systems. However, we plan to prove this assertion in future studies.

We recommend that the GMTI community should take advantage of Model Driven Engineering approach for protocol specification. GMTI format specification should be expressed in a modelling language that is independent of encoding techniques. ASN.1 is one such notation that also offers a number of encoding rules for message serialization, such as BER. We hope to continue our studies in this area provide more information in the near future.

Appendix A
GMTI Specification

This appendix outlines a limited ASN.1 specification of STANAG 4607 Edition 3. We used this model to generate our code for both encoder and decoder applications.

```
-- ASN.1 encoding of STANAG 4607 Edition 3 --
-- Ground Moving Target Indicator Format (GMTIF) --

GMTIF DEFINITIONS
```

```
  AUTOMATIC TAGS EXTENSIBILITY IMPLIED::=
BEGIN

GMTIMessage ::= SEQUENCE
{
  packetHeader  PacketHeader,
  segments    SEQUENCE OF Segment
}

PacketHeader ::= SEQUENCE
{
  version             OCTET STRING (SIZE(2)),
  nationality         IA5String,
  security            PacketSecurity,
  exerciseIndicator   OCTET STRING (SIZE(1)),
  platformID          IA5String,
  missionID           Uint32,
  jobID               Uint32,
  ...
}

PacketSecurity ::= SEQUENCE
{
  classification ENUMERATED
 {
   topSecret   (1),
   secret      (2),
   confidential(3),
   restricted  (4),
   unclassified(5),
   reserved    (6),
   ...
 } DEFAULT unclassified,

  classificationSystem  IA5String,

  code BIT STRING
  {
    noContract     (0), -- bit position 0 is set to 1 --
    orcon          (1),
    propin         (2),
    winintel       (3),
    nationalOnly   (4),
    limdis         (5),
    fouo           (6),
    efto           (7),
    limOffUse      (8),
    nonCompartment (9),
    specialControl (10),
    specialIntel   (11),
    warningNotice  (12),
    relNato        (13),
    rel4Eyes       (14),
    rel9Eyes       (15)
  },
  ...
}

Segment ::= CHOICE
{
  missionSegment   MissionSegment,
  dwellSegment     DwellSegment,
  jobDef           JobDefinitionSegment,
  testAndStatus    TestAndStatusSegment,
  ...
}

MissionSegment ::= SEQUENCE
{
  missionPlan      IA5String,
  flightPlan       IA5String,
  platformType ENUMERATED
  {
    unidentified       (0),
    acs                (1),
    alr-M              (2),
    sentinel           (3),
    rotaryWingRadar    (4),
    globalHawk-Navy    (5),
    horizon            (6),
    e-8                (7),
    p-3C               (8),
    predator           (9),
    radarsat2          (10),
    u-2                (11),
    e-10               (12),
    ugs-single         (13),
    ugs-cluster        (14),
    groundBased        (15),
    uav-Army           (16),
    uav-Marines        (17),
    uav-Navy           (18),
    uav-AirForce       (19),
    globalHawk-AirForce(20),
    globalHawk-Australia (21),
    globalHawk-Germany (22),
    paulRevere         (23),
    marinerUAV         (24),
    bac-111            (25),
    coyote             (26),
    kingAir            (27),
    limit              (28),
    nrl-NP-3B          (29),
    sostart-X          (30),
    watchKeeper        (31),
    ags                (32),
    stryker            (33),
    ags-Hale-UAV       (34),
    sidm               (35),
    reaper             (36),
    warriorA           (37),
    warrior            (38),
    twinOtter          (39),
    other              (255),
    ...
    -- 40 to 254 unused --
  } DEFAULT unidentified,
  platformConfig      IA5String,
  refTime             ReferenceTime,
  ...
}

ReferenceTime ::= SEQUENCE
{
 year   Uint16,
 month  Uint8,
 day    Uint8
}

DwellSegment ::= SEQUENCE
{
  revisitIndex            Uint16,
  dwellIndex              Uint16,
  lastDwellofRevisit      BOOLEAN,
  targetReportCount       Uint16,
  dwellTime               Uint32, -- milliseconds --
  sensorPosition          Position,
  scaleFactor             Scale OPTIONAL,
  senPosUncertainty       SensorPositionUncertainty OPTIONAL,
  sensorTrack             REAL OPTIONAL,
  sensorSpeed             INTEGER(0..8000000) OPTIONAL,
  sensorVerticalVelocity  INTEGER(-128..127) OPTIONAL,
  sensorTrackUncertainty  INTEGER(0..45) OPTIONAL,
  sensorSpeedUncertainty  Uint16 OPTIONAL,
  sensorVerticalVelocityUncertainty Uint16 OPTIONAL,
  platformOrientation     Orientation OPTIONAL,
  dwellArea               DwellArea,
  sensorOrientation       Orientation OPTIONAL,
  mdv                     INTEGER(0..255) OPTIONAL,
  targetReports            SEQUENCE OF TargetReport,
  ...
}

Position ::= SEQUENCE
{
  latitude     REAL,
  longitude    REAL,
  altitude     INTEGER
}

Scale ::= SEQUENCE
{
  latScale  REAL,
  lonScale  REAL
}

SensorPositionUncertainty ::= SEQUENCE
{
  alongTrack   INTEGER(0..1000000),
  crossTrack   INTEGER(0..1000000),
  altitude     Uint16
}
```

```
Orientation ::= SEQUENCE
{
  heading    REAL,
  pitch      REAL,
  roll       REAL
}

DwellArea ::= SEQUENCE
{
  centerLat            REAL,
  centerLon            REAL,
  rangeHalfExtent      REAL,
  dwellAngleHalfExtent REAL
}

TargetReport ::= SEQUENCE
{
  mtiReportIndex       Uint16 OPTIONAL,
  targetLocation       TargetLocation OPTIONAL,
  targetVelocityLOS    Int16 OPTIONAL,
  targetWrapVelocity   Uint16 OPTIONAL,
  targetSNR            Int8 OPTIONAL,
  targetClassification ENUMERATED
  {
    noInfo-LiveTarget          (0),
    trackedVeh-LiveTarget      (1),
    wheeledVeh-LiveTarget      (2),
    rotaryWing-LiveTarget      (3),
    fixedWing-LiveTarget       (4),
    stationaryRotary-LiveTarget(5),
    maritime-LiveTarget        (6),
    beacon-LiveTarget          (7),
    amphibious-LiveTarget      (8),
    person-LiveTarget          (9),
    vehicle-LiveTarget         (10),
    animal-LiveTarget          (11),
    largeMultiReturn-LiveLandTarget (12),
    largeMultiReturn-LiveMaritimeTarger (13),
    -- 14-125 reserved --
    other-LiveTarget           (126),
    unknown-LiveTarget         (127),
    noInfo-SimTarget           (128),
    trackedVeh-SimTarget       (129),
    wheeledVeh-SimTarget       (130),
    rotaryWing-SimTarget       (131),
    fixedWing-SimTarget        (132),
    stationaryRotary-SimTarget (133),
    maritime-SimTarget         (134),
    beacon-SimTarget           (135),
    amphibious-SimTarget       (136),
    person-SimTarget           (137),
    vehicle-SimTarget          (138),
    animal-SimTarget           (139),
    largeMultiReturn-SimLandTarget (140),
    largeMultiReturn-SimMaritimeTarger (141),
    taggingDevice              (143),
    -- 143-253 reserved --
    other-SimTarget            (254),
    unknown-SimTarget          (255),
    ...
  } DEFAULT noInfo-LiveTarget,
  targetClassProbability       INTEGER(0..100) OPTIONAL,
  targetMeasurmentUncertainty  TargetMeasurementUncertainty OPTIONAL,
  truthTag SEQUENCE
  {
    application    Int8,
    entity         Uint32
  } OPTIONAL,
  targetRadarCrossSection    Int8 OPTIONAL,
  ...
}

TargetLocation ::= SEQUENCE
{
  hiResLat         REAL OPTIONAL,
  hiResLon         REAL OPTIONAL,
  deltaLat         Int16 OPTIONAL,
  deltaLon         Int16 OPTIONAL,
  geodeticHeight   INTEGER(-1000..32767) OPTIONAL
}

TargetMeasurementUncertainty ::= SEQUENCE
{
  slantRange               Int16 OPTIONAL,
  crossRange               Int16 OPTIONAL,
  height                   Uint8 OPTIONAL,
  targetRadialVelocity     INTEGER(0..5000) OPTIONAL
}

JobDefinitionSegment ::= SEQUENCE
{
  jobID    INTEGER(1..4294967295),
  sensorID  SEQUENCE
  {
   type  ENUMERATED
   {
     unidentified,
     other,
     hisar,
     astor,
     rotaryWingRadar,
     globalHawkSensor,
     horizon,
     apy-3,
     apy-6,
     apy-8,
     radarsat2,
     asars-2a,
     tesar,
     mp-rtip,
     apg-77,
     apg-79,
     apg-81,
     apy-6v1,
     dpy-1,
     simd,
     limit,
     tcar,
     lsrsSensor,
     ugsSingleSensor,
     ugsClusterSensor,
     imasterGmti,
     anzpy-1,
     vader,
     noStatement(255),
     ...
     -- 28-254 Available for future use --
   },
   model  IA5String,
   ...
  },
  targetFilteringFlag   BIT STRING(SIZE(1)),
  priority        Uint8,
  boundingArea  SEQUENCE
  {
   ptALat  REAL,
   ptALon  REAL,
   ptBLat  REAL,
   ptBLon  REAL,
   ptCLat  REAL,
   ptCLon  REAL,
   ptDLat  REAL,
   ptDLon  REAL,
   ...
  },
  radarMode  ENUMERATED
  {
    unspecified (0),
    mti(1),
    hrr(2),
    uhrr(3),
    hur(4),
    fti(5),
    -- 6-10 available for future use --
    attackControl-SATC(11),
    attackControl(12),
    satc(13),
    attackPlanning-SATC(14),
    attackPlanning(15),
    medResSecSearch(16),
    lowResSecSearch(17),
    wideAreaSearch-GRCA(18),
    wideAreaSearch-RRCA(19),
    attackPlanning-tracking(20),
    attackControl-tracking(21),
    -- 22-30 available for future use --
    wamti-asars-aip(31),
    courseResSearch(32),
    medResSearch(33),
    highResSearch(34),
    pointImg(35),
    smti(36),
    repetitivePtImg(37),
    monopulseCal(38),
    -- 39-50 available for future use --
    search(51),
    emtiWideFrameSearch(52),
    emtiNarrowFrameSearch(53),
    emtiAugmentedSpot(54),
    wamti-asars-2(55),
    -- 56-60 available for future use --
    gmtiPpi(61),
    gmtiExpanded(62),
    nss(63),
    sbs(64),
    wa(65),
    -- 66-80 available for future use --
    grca(81),
    rrca(82),
    sectorSearch(83),
    horizonBasic(84),
    horizonHighSen(85),
    horizonBurnThru(86),
    cresoAcquistion(87),
    cresoCount(88),
    -- 89-93 available for future use --
    wasMtiExo(94),
    wasMtiEndoExo(95),
    ssMtiExo(96),
    ssMtiEndoExo(97),
    -- 98-99 available for future use --
    testStatus(100),
    mtiSpotScan(101),
    mtiArcScan(102),
    hrrMtiSpotScan(103),
    -- 105-110 available for future use --
    grca-globalHawk(111),
    rrca-globalHawk(112),
    gmti-hrr(113),
    -- 114-119 available for future use --
    smallAreaGmti (120),
    wideAreaGmti(121),
    dismountGmti(122),
    hrrGmti(123),
    ...
    -- 124-255 available for future use --
  },
  nominalRevisitInterval    Uint16,
  nominalSensorPositionUncertainty SEQUENCE
  {
   alongTrack      Uint16,
   crossTrack      Uint16,
   altitude        Uint16,
   trackHeading    INTEGER(0..45),
   sensorSpeed     Uint16,
   ...
  },
  nominalSensorValue  SEQUENCE
  {
    slantRangeStdDev   Uint16,
    crossRangeStdDev   REAL,
    targetVelocityLOS  Uint16,
    mdv                Uint8,
    detectionProb      Uint8,
    falseAlarmDensity  Uint8,
    ...
  },
  terrainElevModelUsed  ENUMERATED
  {
    unspecified(0),
    dted0(1),
    dted1(2),
    dted2(3),
    dted3(4),
    dted4(5),
    dted5(6),
    srtm1(7),
    strm2(8),
    dgm50m745(9),
    dgm250(10),
    ithd(11),
    sthd(12),
    sedris(13),
    ...
    -- 14-255 reserved --
  },
  geoidModelUsed      ENUMERATED
  {
    unspecified(0),
    egm96(1),
    geo96(2),
    flatEarth(3),
    ...
    -- 4-255 reserved --
  }
}

TestAndStatusSegment ::= SEQUENCE
{
  jobID        Uint32,
  revisitIndex Uint16,
  dwellIndex   Uint16,
  dwellTime    Uint32,
  hardwareStatus BIT STRING
  {
    spareh (0),
    spareg (1),
    sparef (2),
    calibrationMode (3),
    datalink (4),
    processor (5),
    rfElectronics(6),
    antenna(7)
  },
  modeStatus BIT STRING
  {
      spareh (0),
      spareg (1),
      sparef (2),
      sparee (3),
      temperatureLimitExceeded (4),
      elevationLimitExceeded (5),
      azimuthLimitExceeded (6),
      rangeLimitExceeded (7)
  },
  ...
}

Uint16 ::= INTEGER(0..65535) -- 16-bit unsigned integer --

Uint32 ::= INTEGER(0..4294967295) -- 32-bit unsigned integer --

Int16 ::= INTEGER(-32768..32767)

Int8 ::= INTEGER(-128..127)

Uint8 ::= INTEGER(0..255)

END
```

APPENDIX B GMTI MESSAGE EXAMPLE

This appendix shows the content of a GSER encoded GMTI message from our sample set. This GMTI message contains a packet header, job definition segment and a dwell segment with two target reports. The decoder, $d_{oss}(x)$, produced this message.

```
GMTIMessage ::= {
    packetHeader: PacketHeader ::= {
        version: 31 30
        nationality: CA
        security: PacketSecurity ::= {
            classification: 5
            classificationSystem: CA
            code: 00 00
        }
        exerciseIndicator: 81
        platformID: UAV_JM
        missionID: 0
        jobID: 0
    }
    segments: segments ::= {
        MissionSegment ::= {
            missionPlan: Mission Plan
            flightPlan: Flight Plan
            platformType: 0
        platformConfig: null
        refTime: ReferenceTime ::= {
            year: 2010
            month: 4
            day: 30
        }
    }
    JobDefinitionSegment ::= {
        jobID: 0
        sensorID: sensorID ::= {
            type: 0
            model: null
        }
        targetFilteringFlag: 00
        priority: 1
        boundingArea: boundingArea ::= {
            ptALat: 31.529605865478516
            ptALon: 64.918190002441406
            ptBLat: 31.709268569946289
            ptBLon: 64.917991638183594
            ptCLat: 31.709268569946289
            ptCLon: 65.129173278808594
            ptDLat: 31.529605865478516
            ptDLon: 65.128974914550781
        }
        radarMode: 1
        nominalRevisitInterval: -11856
        nominalSensorPositionUncertainty:
          nominalSensorPositionUncertainty ::= {
            alongTrack: -1
            crossTrack: -1
            altitude: -1
            trackHeading: 255
            sensorSpeed: -1
        }
        nominalSensorValue: nominalSensorValue ::= {
            slantRangeStdDev: 2500
            crossRangeStdDev: 1.99951171875
            targetVelocityLOS: -1
            mdv: -1
            detectionProb: -1
            falseAlarmDensity: -1
        }
        terrainElevModelUsed: 0
        geoidModelUsed: 0
    }
    DwellSegment ::= {
        revisitIndex: 159
        dwellIndex: 0
        lastDwellofRevisit: TRUE
        targetReportCount: 2
        dwellTime: 69438000
        sensorPosition: Position ::= {
            latitude: 31.619480133056641
            longitude: 65.023582458496094
            altitude: 3161
        }
        dwellArea: DwellArea ::= {
            centerLat: 31.651552200317383
            centerLon: 65.061264038085938
            rangeHalfExtent: 4.008196830749512
            dwellAngleHalfExtent: 135.0
        }
        targetReports: targetReports ::= {
            TargetReport ::= {
                mtiReportIndex: 0
                targetLocation: TargetLocation ::= {
                    hiResLat: 31.614080429077148
                    hiResLon: 65.048255920410156
                    geodeticHeight: 31
                }
                targetClassification: 255
            }
            TargetReport ::= {
                mtiReportIndex: 1
                targetLocation: TargetLocation ::= {
                    hiResLat: 31.614080429077148
                    hiResLon: 65.006004333496094
                    geodeticHeight: 31
                }
                targetClassification: 255
            }
        }
    }
  }
}
```

Appendix C
Cost Analysis of Generated Code

If we were to implement a GMTI encoder and decoder code by hand, how many developers do we need and long will it take? This appendix tries to answer these questions. To help us answer these questions, we used a well known software cost estimation model, COCOMO 81 Intermediate Mode [11], for our level of effort analysis.

Usually, when doing a software cost estimate, the number of lines of code of the final product is unknown and is estimated based on number of Function Points. In our analysis, the ASN.1 compiler provided us the exact number of lines of code. Therefore, to calculated level of effort, we first count the number of lines of code that was generated by an ASN.1 compiler. Let $SLOC$ be the number of single line of code (SLOC) that a compiler generated from an GMTI model specified in Appexdix A, excluding blank lines.

$$SLOC = 2,284 \tag{1}$$

The compiler produces code in the C programming language. Let G be the G Factor when using C programming language.

$$G = 128 \tag{2}$$

Let EAF be the Effort Adjustment Factor that is calculated in table 9.

$$EAF = 1.14492 \tag{3}$$

For this analysis, we selected Development Mode to be Organic instead of Embedded or Semidetached. For Organic Model, $a = 3.2$ and $b = 1.05$. Therefore, the APM, Adjusted Person Months is

$$APM = a(SLOC/1000)^b(EAF) = 8.78 \tag{4}$$

APM is the number of person months required to implement GMTI encoder and decoder by hand.

Using Organic Mode, let $c = 2.5$ and $d = 3.8$. Let $TDEV$ be the development time in months required to build GMTI encoder and decoder by hand.

$$TDEV = c(APM)^d = 2.5(8.78)^{0.38} = 5.01 \tag{5}$$

Let NP be the number of personnel required to build a GMTI encoder and encoder manually. This tells us how many developers are required to work on this project.

$$NP = APM/TDEV = 8.78/5.01 = 1.75 \tag{6}$$

In summary, if we uses ASN.1 compiler to generated C programming language code for GMTI message serialization, we would save 1.75 developers and 5.01 months in development effort.

TABLE 9
COCOMO Cost Drivers and their values used to derive EAF Effort Adjustment Factor.

Cost Drivers	Values
Required Reliability	1.4 (Very High)
Database Size	0.94 (Very Low)
Product Complexity	1 (Nominal)
Execution Time Constraint	1 (Nominal)
Main Storage Constraint	1 (Nominal)
Virtual Machine Volatility	1 (Nominal)
Computer Turn Around Time	0.87 (Low)
Analyst Capability	1 (Nominal)
Application Experience	1 (Nominal)
Programmers Capability	1 (Nominal)
Virtual Machine Experience	1 (Nominal)
Language Experience	1 (Nominal)
Modern Programming Practice	1 (Nominal)
Software Tools	1 (Nominal)
Required Development Schedule	1 (Nominal)
Effort Adjustment Factor	1.14492

ACKNOWLEDGMENTS

I would like to thank George Dinardo for providing feedback and his expert knowledge on STANAG 4607.

REFERENCES

[1] STANAG 4607 JAS (Edition 3), *NATO Ground Movig Target Indicator (GMTI) Format*. NSA, 14 September 2010
[2] O. Dubuisson, *ASN.1. Communication between Heterogeneous Systems*, OSS Nokalva, 5 June 2000.
[3] J. Larmouth, *ASN.1 Complete*. OSS Nokalva, 1999
[4] ISO 8824-1 — ITU-T X.680, *Specification of Basic Notation*. ITU-T, November 2008.
[5] ISO 8825-1 — ITU-T X.690, *BER, CER, and DER*. ITU-T, November 2008.
[6] ISO 8825-5 — ITU-T X.694, *Mapping W3C XML Schema Definitions into ASN.1*. ITU-T, November 2008.
[7] SMPTE 336M-2003, *Data Encoding Protocol Using Key-Length-Value*. White Plains, NY: 22 August 2007.
[8] D. Crocker, Ed. Brandenburg InternetWorking, P. Overell, *RFC 5234, Augmented BNF for Syntax Specification: ABNF*. IETF, January 2008.
[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *RFC 2616, Hypertext Transfer Protocol: HTTP/1.1*. IETF, June 1999.
[10] J. Klensin, *RFC 2821, Simple Mail Transfer Protocol*. IETF, April 2001.
[11] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[12] T. Bray, C. Frankston, A. Malhotra, *Document Content Description for XML*. W3C, 31 July 1998.
[13] J. Cowan, R. Tobin, *XML Information Set (Second Edition)*. W3C, 4 February 2004.
[14] J. Schneider, T. Kamiya, *Efficient XML Interchange (EXI) Format 1.0*. W3C, 10 March 2011.
[15] Unified Modeling Language (UML), http://www.uml.org/
[16] Cygwin, http://cygwin.com/
[17] Open Source Software (OSS) ASN.1 Compiler, http://lionet.info/asn1c/blog/
[18] ASN Lab ASN.1 Compiler, http://www.asnlab.com/

James McAvoy, CD, BSc Contracting to DLCSPM 4-PMO ISTAR C2 as a Senior ISR Systems Engineer for the last two years developing various applications to enhance tactical ISR capabilities.