NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL AWARENESS MODULE

by

Matthew S. Bernard

June 2019

Thesis Advisor:	Gurminder Singh
Co-Advisor:	Alan B. Shaffer

Approved for public release. Distribution is unlimited.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2019	**3. REPORT TYPE AND DATES COVERED** Master’s thesis

4. TITLE AND SUBTITLE MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL AWARENESS MODULE	**5. FUNDING NUMBERS**
6. AUTHOR(S) Matthew S. Bernard	
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**

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.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The purpose of this research was to develop a methodology for creating cyber situational assessment between a physical network and its virtual copy to enable cyber situational awareness (CSA) in the Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) Awareness Module. The research focused on the nodal attributes of a network that contribute most toward creating CSA while also lending to virtualization. This work describes a conceptual system design and toolset that enables users to collect, normalize, and store network attributes in a non-relational database, and analyzes open-source software that can be used as its intermediary build platform. We also describe a conceptual design for the MAVNATT Awareness Module, which queries, selects, and stores the nodal attributes in a relational database schema for tracking and management. The relational schema in this work was written in MySQL and reverse engineered using Oracle’s SQL Developer to check for correctness. Documentation of the schema tables and their relationships, cross-referenced with a matrix of the attributes that can be virtualized in VirtualBox, accompany the provided SQL code.

14. SUBJECT TERMS MAVNATT, network administrator training, cyber situational awareness, cyber situational assessment, SIEM, relational schema, awareness module, awareness, VirtualBox, MySQL, OSSIM	**15. NUMBER OF PAGES** 125
	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

MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL AWARENESS MODULE

Matthew S. Bernard
Lieutenant, United States Navy
BA, San Diego State University, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2019

Approved by: Gurminder Singh
Advisor

Alan B. Shaffer
Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The purpose of this research was to develop a methodology for creating cyber situational assessment between a physical network and its virtual copy to enable cyber situational awareness (CSA) in the Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) Awareness Module. The research focused on the nodal attributes of a network that contribute most toward creating CSA while also lending to virtualization. This work describes a conceptual system design and toolset that enables users to collect, normalize, and store network attributes in a non-relational database, and analyzes open-source software that can be used as its intermediary build platform. We also describe a conceptual design for the MAVNATT Awareness Module, which queries, selects, and stores the nodal attributes in a relational database schema for tracking and management. The relational schema in this work was written in MySQL and reverse engineered using Oracle's SQL Developer to check for correctness. Documentation of the schema tables and their relationships, cross-referenced with a matrix of the attributes that can be virtualized in VirtualBox, accompany the provided SQL code.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AWS	Amazon Web Services
BTU	British thermal unit
CANES	Consolidated Afloat Networks and Enterprises Services
CPU	central processing unit
CSA	cyber situational awareness
CSAP	Cyber Situational Awareness Pyramid
Ctrl	controller
DNS	Domain Name System
eth	ethernet
GB	gigabyte
GPU	Graphics Processing Unit
GraphML	Graph Markup Language
hr	hour
HTTP	Hypertext Markup Language
I/O	input / output
IDE	integrated development environment
IDS	intrusion detection system
in	inches
IP	Internet Protocol
JDL	Joint Director of Laboratories
kVA	kilovolt-ampere
lb	pound
LPT	parallel port interface
MAC	media address control
MAVNATT	Mapping, Awareness, and Virtualization Network Administrator Training Tool
MB	megabyte
NAT	network address translation
NIC	network interface card
OS	operating system

OSSIM	Open Source SIEM
RAM	random access memory
RDBMS	relational database management system
SA	situational awareness
SIEM	security information and event manager
SQL	Structured Query Language
STDIN	standard input
TB	terabyte
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
USM	Unified Security Management
VBox	VirtualBox
VM	virtual machine
VRAM	video random access memory
WUI	web-based user interface
XML	EXtensible Markup Language

ACKNOWLEDGMENTS

This work is dedicated to the love of my life, Mrs. Gerna Bernard, without whom none of this would have been possible. You have selflessly supported me every single day of the last 20 years of my career, patiently waiting for me to return from 12-hour shifts or time at sea while tirelessly raising our two children. And now that I am on land, you find yourself having to share me with homework and thesis writing. Your sacrifice allowed this work (and all others!), and your love gave me the resolve to finish. Thank you, baby!

To my mentors Dr. Singh and Dr. Shaffer: thank you both for allowing me to talk through ideas in your office or via email and for helping me to navigate this project. Your encouragement and feedback through each step of the process helped me move this project forward.

And to my steadfast pack mates Doran, Jason, and Aaron: you always had my back, provided an ear, practical advice and context, helped with problem bounding, and offered copious amounts of encouragement. Thank you for patiently waiting as I geeked out before accomplishing actual work and for being there regardless of the late hour or last-minute press to finish. You guys made the arduous path worth remembering. A-Whoooo!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) is a system created at the Naval Postgraduate School (NPS) to solve the problem of network administrators training. It is to the network administrator's advantage to train on the very network that they manage. Yet there are consequences to making mistakes on a tactical network, such as cutting off communications accidentally, disabling critical parts of the system, and allowing unrestricted access where it should be controlled. This possibility makes it dangerous for tactical commanders to allow administrators to train on their operational networks. MAVNATT solves this problem by making a virtual copy of the network, providing a realistic experience for the network administrator. The virtualization is meant to accurately replicate the network that the administrator manages, allowing them to train on their network without consequence (McBride, 2015).

MAVNATT uses three separate modules—Mapping, Awareness, and Virtualization—to conduct different tasks. To make this system more useful, the Awareness Module is meant to assist the network administrator and the trainer by presenting them with the situational awareness of the network. This is to allow them to see activity within their operational network, the virtual network, and the differences between the two.

This research seeks to expand on the original requirements of MAVNATT by defining Cyber Situational Awareness (CSA) in this context and to find an effective and efficient way to achieve CSA for the tactical operator and trainer. The following research questions are investigated:

1. Primary Question

How can CSA of a tactical network be achieved?

2. Secondary Question

What kind of information needs to be collected, and how should it be analyzed and visualized to produce effective CSA?

A. OBJECTIVES

There are two main objectives for this work. The first objective is to determine an efficient method to provide the tactical network administrator and trainer with CSA. The second objective is to implement a module for MAVNATT that provides baseline configurational awareness to the network administrator.

B. BENEFITS OF STUDY

Cyberspace is a man-made environment which requires the cyber operator to constantly train in order to maintain proficiency with regard to their tasking. For defensive cyber operators, CSA is critical to the defense of their cyber terrain, as a network cannot be defended without knowing how the adversary might act upon or is moving within that network. CSA begins with having a working understanding of the cyber terrain and is complete when the current processes within that terrain are known.

Currently, tactical network administrators are not considered defensive cyber operators because they often do not receive the training, experience, or even the tools to defend their networks. Yet tactical network administrators are the first line of defense for their networks and they need to become defensive cyber operators for their terrain. This work benefits the Department of Defense and the Department of the Navy by advancing MAVNATT with a design for its Awareness Module and its infrastructure, which will ultimately provide tactical network administrators with a tool that they can use to gain CSA of their cyber terrain and train without adversely affecting that terrain.

C. ORGANIZATION

The remainder of this thesis is organized in the following manner:

The second chapter, “Background and Previous Work,” outlines the previous work that has been done with situational awareness and how it can be applied to computer networks. The chapter then reviews three different Security Information and Event Manager (SIEM) systems, and how they contribute to network administrator situational awareness. We end the chapter with an overview of MAVNATT itself.

Chapter III, “Awareness Module Design,” describes a conceptual model for the new Awareness Module in terms of the original MAVNATT goals to include information and process flows. Also described is a design for the MAVNATT system infrastructure and how OSSIM can be used an intermediary platform to reduce MAVNATT’s development time.

The next chapter, “MAVNATT Relational Schema Implementation,” discusses the types of awareness attributes with regard to training, the selection of the various attributes of a node that can be virtualized, the relational schema design considerations, and the schema implementation.

The thesis ends with “Conclusion and Future Work,” which covers the accomplishments and limitations of the research. The future work section describes possible improvements and refinements to the MAVNATT system.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND PREVIOUS WORK

A. SITUATIONAL AWARENESS

According to Tadda & Slaerno (2010), situational awareness (SA) "is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future to enable decision superiority" (p. 17). This expands on Endsley's (1995) original definition that SA is the understanding or "state of knowledge" that an actor has when he knows what is going on around him. SA is what people use as a base for decision making when proposing an action to take within the near future. Endsley (1995) states that SA is created via a phased hierarchy of escalating knowledge through three discrete levels (p. 36).

Endsley's (1995) first level of SA knowledge is "the *perception* of the elements in the environment within a volume of time and space" (p. 36). The volume of time and space considered for an environment provides a context by which to determine what elements will be influencing factors to a situation. For example, a person in a daylight street-crossing situation would look for anything that could potentially intersect the path he/she is considering taking across that street. To decide if it is safe to cross the street, the presence of cars are influencing factors and the clouds in the sky are not.

The second level of knowledge from Endsley's (1995) definition is the *comprehension* of the meaning of those elements. Referring back to the previous example, just because cars are present on the street does not mean that it is not safe to cross. The speed and direction of travel of the cars present give meaning to how much they influence the safety of the situation.

The final level of knowledge from Endsley's (1995) SA definition is the *projection* of the status of those elements in the near future. Referring again to the street-crossing situation, the person will project the speed and the direction of the cars present to see if they could intersect his/her intended path during the crossing window.

The person, detecting the elements relevant to the situation, comprehending their meaning, and projecting their status in the near future, is now fully aware of the crossing situation. This state of knowledge enables a fourth level of knowledge that McGuinness and Foy call *resolution*, which enables decision makers to reach a decision on what action to take (as cited in Tadda & Salerno, 2010, p. 19). The person in the street-crossing example now has the information required to resolve whether a safe crossing is possible and to act accordingly.

Endsley (1995) notes that SA does not take into account all of a person's knowledge and it only refers to the knowledge of the "state of a dynamic environment" (p. 36). The hierarchical model of SA can be thought of as a two-part process where in the first part information is gathered for processing within a second, cognitive process. Endsley (1995) defines *situation assessment* to refer to both parts as a "process of achieving, acquiring, or maintaining situation awareness" (p. 36). Tadda and Salerno (2010) summarize SA as being a cognitive process that takes place in the minds of humans as it is a state of knowledge and note that situation assessment is a technical "process or set of processes [that] lends itself to automated techniques" (p. 16).

1. Cyber Situational Awareness

Cyber is defined as being "of, relating to, or involving computers or computer networks" ("Cyber," 2018). Cyber Situational Awareness (CSA) is "a subset of SA which concerns the cyber environment" and includes awareness of interesting or suspicious activity of that environment (Franke & Brynielsson, 2014, p. 20). CSA is very useful to stakeholders of computer networks as it helps them to determine and maintain the security of those networks (Franke & Brynielsson, 2014, p. 19). Since CSA is a subset of SA, it follows that the processes of CSA are subsets of SA. The production of CSA, like SA, takes place via a two-part process where information is first gathered, and then refined by computer systems for presentation to the network's stakeholder. The stakeholder fuses that information with other knowledge gained from outside the network via cognitive processing to create CSA. Since cognitive processing produces CSA and takes place within the mind, computers cannot attain CSA. However, they can gather the necessary information from the network, refine it, and present it to stakeholders to enable them to achieve CSA (Tadda & Salerno, 2010, p. 16).

2. Cyber Situational Assessment

The initial process of creating CSA is *cyber situational assessment*, defined here as "the process of collecting and refining cyber data to enable the acquisition or maintenance of cyber situation awareness" (Endsley, 1995, p. 36; Tadda & Salerno, 2010, p. 16). To achieve cyber situational assessment, it is necessary to first identify the information to perceive based on the context of the network in use. Table 1 lists examples of entities and data sources that provide elements required for this purpose. In general, Giacobe notes that fixed observable data can come from any cyber security sensor or other routine IT-support data source, such as network-based IDS, host-based IDS, host log file data, baseline configuration information to include operating systems and patches installed, and physical system locations. In addition to these data sources, stakeholders also collect soft data from sources external to the network such as user observations and cyber intelligence in order to generate cyber situation assessment (Giacobe, 2010).

The next step in cyber situation assessment is to give meaning to the individual elements perceived. The elements' meanings are then fused together to understand the state of the situation (Endsley, 1995; Giacobe, 2010). To perform this data fusion, Giacobe suggests using the Joint Director of Laboratories (JDL) Data Fusion Model, depicted in Figure 1. The JDL Data Fusion Model provides a framework for combining various sources of data to create a better understanding of an observed situation by using algorithms to combine the data and infer the meaning of the data in context at each level of data fusion (Giacobe, 2010). The JDL model defines six data fusion levels, as follows:

- Level 0: Source Pre-Processing normalizes and aligns source data to prepare it for analysis (Giacobe, 2010).
- Level 1: Object Refinement combines the multi-source "data to identify individual security events" (Giacobe, 2010, p. 2).
- Level 2: Situation Refinement "combine[s] multiple individual entities to provide a current-state system perspective" (Giacobe, 2010, p. 2).
- Level 3: Threat Refinement "provide[s] [the] capability to predict future states of the system" (Giacobe, 2010, p. 2).

- Level 4: Process Refinement "addresses the system's capability to task sensors and maintain their [rules]", refining the previous processes (Giacobe, 2010, p. 2).
- Level 5: Cognitive Refinement / Human Computer Interface is how the human analyst incorporates information from the data fusion system (Giacobe, 2010).

Table 1. Examples of Entities and Source Data. Adapted from Giacobe (2010).

Entity	**Source Data Examples**
Defended Host	• Physical Location • Hardware Details • Operating System • List of Patches Applied • Application Level Software Installed • CPU and Memory Utilization • Applications Currently Running • End-User Account Access Data • Security Log Data • Classification of Data Stored on the System
Intrusion	• IDS Type and Location • Source Address • Destination Address • Attack Method • Time of Attack
Attacker	• Source Address(es) • Methods of Attack
Flows of Data	• Source and Destination Addresses • Type of Traffic (Protocols Used) • Traffic Volume • Encryption Used

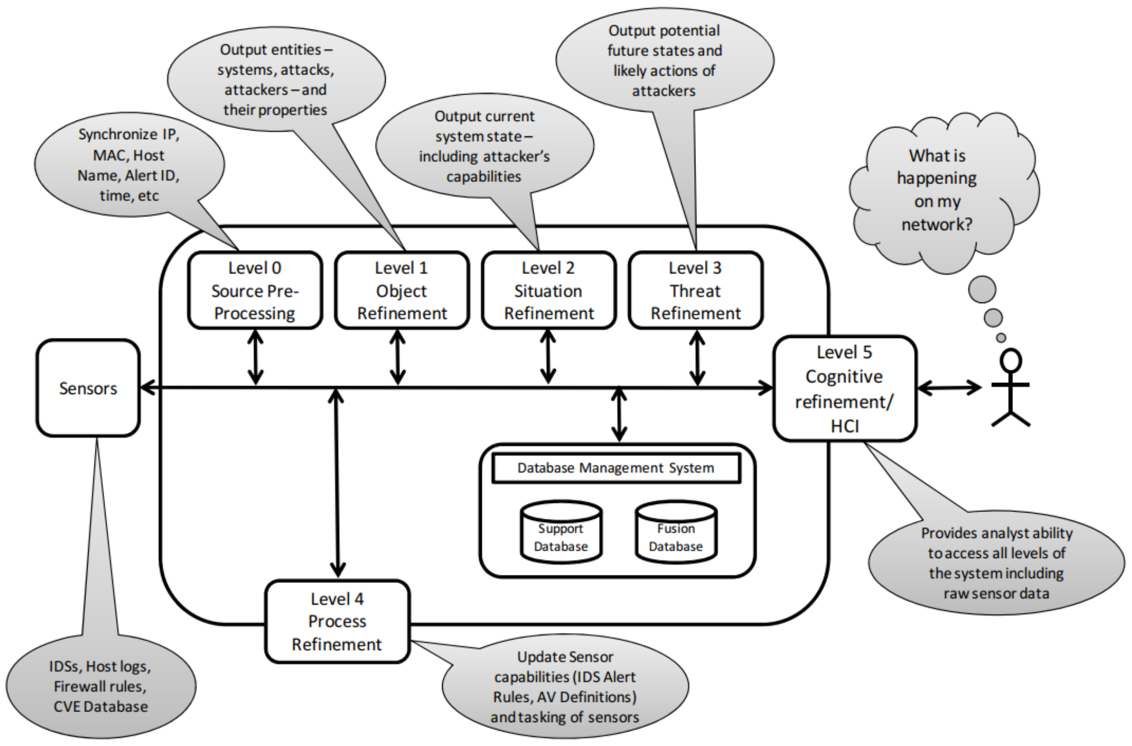

Output entities – systems, attacks, attackers – and their properties
Output potential future states and likely actions of attackers
Synchronize IP, MAC, Host Name, Alert ID, time, etc
Output current system state – including attacker's capabilities
What is happening on my network?
Level 0 Source Pre-Processing
Level 1 Object Refinement
Level 2 Situation Refinement
Level 3 Threat Refinement
Sensors
Level 5 Cognitive refinement/ HCI
Database Management System
Support Database
Fusion Database
Provides analyst ability to access all levels of the system including raw sensor data
Level 4 Process Refinement
IDSs, Host logs, Firewall rules, CVE Database
Update Sensor capabilities (IDS Alert Rules, AV Definitions) and tasking of sensors

Figure 1. Joint Director of Laboratories Data Fusion Model. Source: Giacobe (2010).

B. SECURITY INFORMATION AND EVENT MANAGERS

As discussed in the previous section, cyber situational assessment requires the fusion of diverse sources of fixed observable network data. Security Information and Event Managers (SIEMs) are software suites that provide a range of capabilities for this process. According to one current market definition, SIEMs are "defined by the customer's need to analyze event data in real time for the early detection of targeted attacks and data breaches, and to collect, store, analyze, investigate and report on event data for incident response, forensics and regulatory compliance" (Kavanagh & Bussa, 2017).

1. Principles of a SIEM

According to Holik, Horalek, Neradova, Zitta, & Marik (2015) in their article reviewing SIEM features and principles, a SIEM solution must contain or provide a source of fixed observable data, a log collection service, normalization of the logs and data collected, both a rule and correlation engine, log storage, and system monitoring. Figure 2 depicts the architecture of these components.

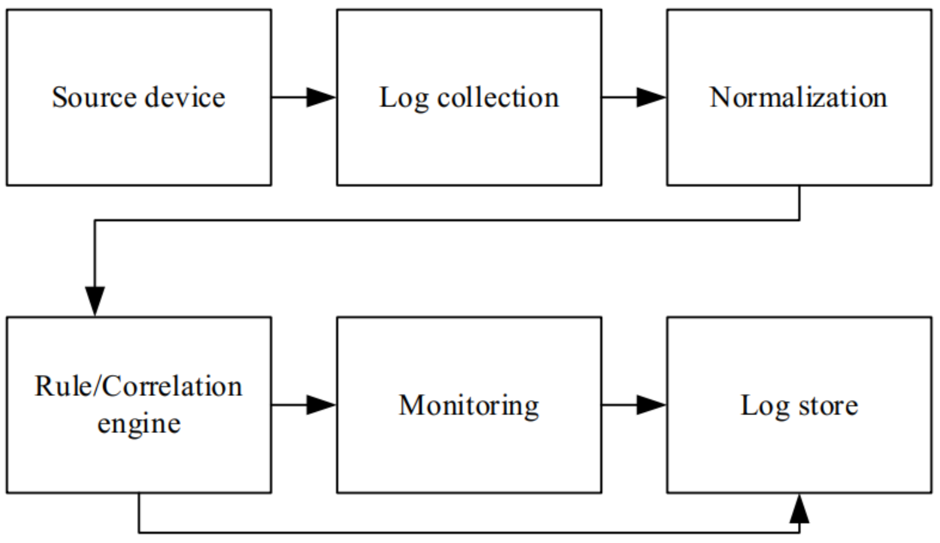


Figure 2. SIEM Architecture. Source: Holik et al. (2015).

Normalization of the logs and data must be provided so that incoming data can be converted into a common format to enable its analysis (Bhatt, Manadhata, & Zomlot, 2014, p. 36). A rule and correlation engine then reacts to and correlates the normalized data using various algorithms ranging from expert systems to neural networks (Giacobe, 2010).

SIEMs can be open source or commercial, and offer a wide range of capabilities depending on security requirements and budget constraints. They generally incorporate or have the ability to interface with networking tools such as network mappers, firewalls, vulnerability scanners and analyzers, intrusion detection and prevention systems, configuration compliance solutions, and databases (Holik et al., 2015). While their architectures are generally the same, SIEMs do not use the same tools and techniques, and may serve niche areas or provide a less robust set of tools (Nicolett & Kavanagh, 2011). Due to the differences in features and pricing, network administrators may need to use a combination of SIEMs and tools to cover their security requirements.

2. SIEM Examples

This section presents three examples of open source SIEMs to demonstrate the wide variety of technologies that can be used to create a cyber situational assessment. The first is Apache Metron, which employs Big Data tools and is fully scalable to accommodate large industrial applications. The second is SIEMonster, a SIEM that seeks to deliver a full range of security capabilities to end users. The third is OSSIM, an open source version of Unified Security Management, AlienVault’s commercial product.

a. Apache Metron

According to Vetticaden (2016), “Apache Metron is a cyber security application framework that provides organizations the ability to ingest, process and store diverse security data feeds at scale in order to detect cyber anomalies and enable organizations to rapidly respond to them.” Metron was released in 2016, and the current release is 0.4.1. Metron has prebuilt installations for CentOS 6 and Ubuntu 14.04, while instructions for deployment to other platforms such as Amazon Web Services (AWS) are available (Foley, 2017; Berman, 2018; Vetticaden, 2017).

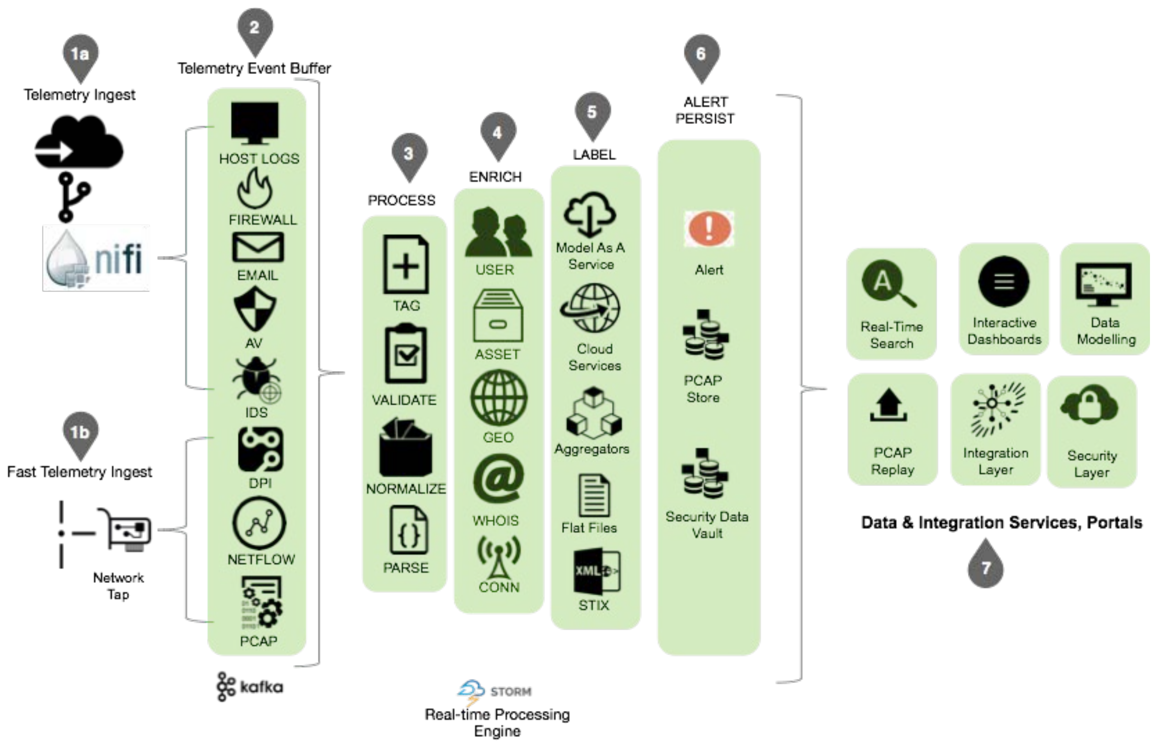


Figure 3. Apache Metron Logical Components. Source: Vetticaden (2016).

Metron's scalability is achieved through the use of a Big Data toolset and can be logically broken down into four components. The first is telemetry ingestion, which can take two paths. One path uses Apache NiFi (Figure 3, 1a), which can be run as a single Java Virtual Machine instance, or in a cluster to fit Metron's deployment scale. NiFi is used to manage the incoming flow of data from the network (Apache Software Foundation, n.d.). In Metron, NiFi receives network endpoint generated logs to include hosts, firewalls, intrusion detection systems, and antivirus platforms. The other path of telemetry ingestion is via network tap to collect raw network packet data (Figure 3, 1b) (Vetticaden, 2016).

Both of these paths flow into Metron's second component, which is Apache Kafka (Figure 3, 2), a distributed message buffering system that organizes the telemetry into different "topics." Like NiFi, Kafka can deploy as a cluster to meet the Metron's deployment scale. Each server within the Kafka cluster can have one or more topic partitions, which act as a buffer that maintains the log messages in the order they were received. The log messages in each topic are replicated across a desired number of topic partitions so that the logs can be consumed by Metron in a parallelized way (Sookocheff, 2015).

The third logical component of Metron handles message processing (Vetticaden, 2016), which is done by Apache Storm, “a task parallel, open source distributed computing system” (DeZyre, 2015, para. 10). Storm processes incoming message streams through a *topology*. The topology starts with the Kafka cluster that the information is read from and progresses to all of the various “bolts” along the intended path of the topology (Soni, 2018). These bolts in the topology are the processes required to parse and normalize (Figure 3, 3), enrich (Figure 3, 4), and label (Figure 3, 5) the incoming messages. The enrichment of the message involves adding metadata such as the geographic locations of the traffic’s source and destination to the log itself. Any threat intelligence that correlates to the message is also added to the log and the message becomes “labeled.” Any telemetry that meets criteria will then initiate an alert (Figure 3, 6), which is indexed in Elasticsearch and stored in a Hadoop Distributed File System for analysis.

The final logical component of Metron is its user interface (Figure 3, 7), which provides real-time searches, security dashboards, data modeling and engineering, and system access to modify and customize NiFI, Kafka, and Storm settings (Vetticaden, 2016).

b. SIEMonster

SIEMonster is a fully scalable SIEM framework that is based on a comprehensive collection of open source software, which provides nearly every service a SIEM can have (Table 2), although a paid version provides advanced correlation and reporting capabilities. The current version of SIEMonster is 3.0 and uses the CoreOS operating system, which is designed for clustered deployments. CoreOS supports Docker, which provides an alternative to using virtual machines. Docker instead uses containerized space to fully host applications that may even be built for other platforms. When combined, CoreOS and Docker provide SIEMonster the ability to deploy as many nodes and clusters as necessary so as to properly distribute the computing load across SIEMonster’s servers. SIEMonster can be deployed in a variety of ways including bare metal, virtual machine instances, or on

AWS. For a corporate deployment to a virtual AWS instance, the recommended hardware for each of the five servers is an eight-core CPU, 32GB of RAM, with solid-state drives for storage. A deployment of SIEMonster on five of these servers can monitor an enterprise network generating 47 million logged events per hour and can be scaled upwards as necessary (Rock & Bycroft, 2018).

SEIMonster's six primary functions are interwoven among five different servers: Proteus, Makara, Capricorn, Kraken, and Tiamat. The six functions include log collection, log processing, visualization, vulnerability detection and reporting, network user ticketing to report problems, and open source intelligence integration. The Makara server provides system management via its web application front end and serves as a centralized backup server for all container configuration data within SIEMonster. Each of the five servers has the ability to monitor their Docker containers for usage and performance. When a compute load imbalance exists between the servers, Makara provides system orchestration and moves containers between clusters to equalize resource usage (Rock & Bycroft, 2018).

Cyber situational assessment within SIEMonster begins by collecting netflow traffic and logs from the network nodes and endpoints it has been directed to monitor, converting said traffic and logs into JSON files, and sending them to the RabbitMQ message queue, which takes place in the Proteus server. Makara accepts messages from Proteus using a Logstash indexer to parse and normalize logs prior to sending them to the Elastic clients for the real-time search and analytics capabilities that reside on Proteus and Capricorn. The last two servers, Kraken and Tiamat, are Cluster Nodes 1 and 2, respectively, and employ Elasticsearch to provide long-term SIEM data storage redundancy (Rock & Bycroft, 2018).

Logstash is a data collection and manipulation engine, and has the ability to accept log input from a wide variety of sources such as TDP/UDP, files, Windows, STDIN, and many others. Logstash has the ability to filter and manipulate messages, which is used to parse and normalize log data coming into SIEMonster to ensure data uniformity. Logstash's ability to output data is just as powerful as that of what it can accept. It supports many output options including HTTP, TCP/UDP, files, emails, and online services (Turnbull, 2013).

Elasticsearch is a distributed non-relational database, thus it has no schema or logical structure (Kononenko, Baysal, Holmes, & Godfrey, 2014). Elasticsearch uses an inverted index to store key-value pairs. In the case of SIEMonster, the key is a term from a JSON document and the value is a reference to the JSON document itself. The searches that can be performed in Elasticsearch depend on the indexes that have been generated. A search term has to match at least part of the beginning of a term in the index, if not the whole term, in order to have a successful result returned. The index most often created uses full words from the text of the document. Therefore, the full word index can also be used to find substrings that start with the beginning of the term. A second index containing the terms spelled backwards provides the ability to find substrings starting at the end of the terms. Custom indices can be created to enable searches based on the data received and the desired search results (Brasetvik, 2013).

Table 2. SIEMonster v3.0: List of Open Source Software with Purpose. Adapted from Rock and Bycroft (2018).

Purpose	**Software**
Log Retrieval	Filebeat, Syslog, NXlog, OSSEC
Process	RabbitMQ, Logstash, Kustodian
Intrusion Detection	OSSEC w/ Wazuh fork
Rules, Storage, Alerting	Logstash, Elasticsearch, 411, Kustodian, OSSEC Wazuh fork
Security	Search Guard, Firewalls & Lockdowns
Audit and Discovery	Open AudIT by Opmantek
Threat Intelligence	MineMeld, opensource feed aggregation
Visualization	Kibana, Slack, Dradis IDE
Backup scripts, Maintenance	Kustodian scripted AWS Glacier backup, archive, & restore
Vulnerability Scanning	OpenVAS, Dradis IDE
Ticketing	411/FIR Incident Response

c. *Open Source SIEM*

Open Source SIEM (OSSIM) is a free, open source version of AlienVault's Unified Security Management (USM) system (Geil, 2017), in development since 2003 (Lkhamsuren, 2012). OSSIM has most of the security capabilities of USM, to include asset discover and inventory, vulnerability assessment, intrusion detection, behavioral monitoring, and SIEM event correlation. Support is available via AlienVault's product forums (AlienVault, 2017), and it can also receive and contribute information to AlienVault Open Threat Exchange, which is an "open threat information sharing and analysis network" (AlienVault, n.d.). According to a 2015 comparison of three open source SIEMs, OSSIM included nine native sensors and is compatible with nearly 2000 others. It excelled in the categories of presentation and investigation, but has more hardware requirements and is more complex to configure and install than other SIEMs (Leszczyna & Wróbel, 2015).

OSSIM has several limitations that discourage greater adoption of its use. The behavioral monitoring is not as robust as it is in USM, since OSSIM does not provide pre-configured correlation rules. While the capability does exist, it is up to the administrator to create the rules, and no professional support is provided (AlienVault, 2017). Additionally, OSSIM does not provide functionality for log management. When used professionally, the installer must provide an external logging solution (kcoe, 2017a). All of these limitations are due to the fact that OSSIM is an open source version of AlienVault USM, and is not meant to be fully capable (Lkhamsuren, 2012). It has been shown that OSSIM can be successfully integrated with a validated forensic storage solution to overcome the lack of an incorporated log management solution (Afzaal, Sarno, Dantonio, & Romano, 2012). Though OSSIM is free to obtain, the configuration and deployment costs can be significant (AlienVault, 2017).

OSSIM is designed to be deployed only as a single server, and is most appropriate for low-volume deployments that do not require federation (AlienVault, 2017). Though OSSIM may not be configured to automatically handle federation, work exists which documents how federation involving multiple instances of OSSIM can be accomplished (Aguirre & Alonso, 2012). The server installation comes with a built-in sensor package, but performance will improve in a larger network that has additional sensors (AlienVault, 2017). According to Madrid et al. (2009), OSSIM uses a distributed architecture with four basic elements. The first is the information capture elements, which are plugins that interface with network security sensors to acquire fixed observable data from the network. Second are the databases, which "collects all events received from the [sensors and] the alarms generated by the server's correlation engine" (Madrid et al., 2009, p. 2). The third element is a server, which "correlates events in the database in order to detect patterns indicating system vulnerabilities or attacks" and additionally works to remove "most false-positive alerts" (Madrid et al., 2009, p. 2). Finally, the fourth element is a web-based management console that provides a "graphical front-end to the system [which] allows administrator[s] to browse through reports, alarms, and statistics" (Madrid et al., 2009).

AlienVault no longer publishes dedicated documentation for OSSIM. Many articles and papers that have previously referenced OSSIM documentation point to the domain "www.ossim.net." This site has been removed and all documentation for OSSIM is now covered under USM Appliance documentation, though none of it will specifically address OSSIM (website: https://www.alienvault.com/documentation/usm-appliance.htm) (kcoe, 2017b).

Since OSSIM is a reduced version of USM, the USM Appliance documentation serves both SIEMs even though it only references the commercial product. A typical deployment of OSSIM in a professional application would be likely to occur if the size of the network was relatively manageable, for example, 100 assets. A deployment for this scenario would require only one server to host all four architectural elements. Though not specifically written for OSSIM, it is reasonable to assume that any hardware requirements for the AlienVault USM Appliance All-In-One server will apply to OSSIM as well. Table 3 details the hardware specifications that would support a network of 150 assets with the following device performance:

- 200 Million Events in Database (Maximum)
- 1000 Events Per Second Data Collection Rate
- 1000 Events Per Second Data Correlation Rate
- 100 Megabits Per Second IDS Throughput (AlienVault, 2018)

Table 3. OSSIM Hardware Specifications. Adapted from AlienVault (2018).

Hardware Specifications	
Form Factor	1U
Length x Width x Height (in)	26.6 x 17.2 x 1.7
Weight (lb)	42
Power Supply	2 x 700 / 750W
Network Interfaces	6 x 1GbE
CPU	2x Intel Xeon E5620 2.5GHz (8 Cores)
Storage Capacity (TB) Compressed / Uncompressed	9.0/1.8
Disk Array Configuration	RAID 10
Memory (GB)	24
Redundant Power Supply	Yes
IPMI Interface	Yes
Max Heat Dissipation (BTU/hr)	2,794.54
Max Power Consumption (kVA)	0.837

C. MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL

The Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) provides training and evaluation to tactical network administrators. Berndt (2016) explains that "MAVNATT is a [lightweight] system that creates a fully partitioned, stateful, and virtualized instance of an operational network" (p. 5) with all of its hosts and components. This allows network administrators to train on the virtualized copy of their network, and removes network-training risks to the tactical unit's communications. MAVNATT uses three different modules to accomplish this (McBride, 2015, p. 37–40).

MAVNATT begins the virtualization process from a target network baseline, modeled as an open-format graph file. The Mapping Module verifies the network baseline by using NMAP to scan the network for element discovery, and then compares the results to the baseline (Collier, 2016). The Awareness Module then enhances the resulting network map by collecting additional details from the network elements, passing the completed map to the Virtualization Module.

Figure 4 depicts the interaction between the MAVNATT modules. Greater detail on the operations of the Mapping and Virtualization modules can be found in Berndt's and Collier's individual theses (Berndt, 2016; Collier, 2016).

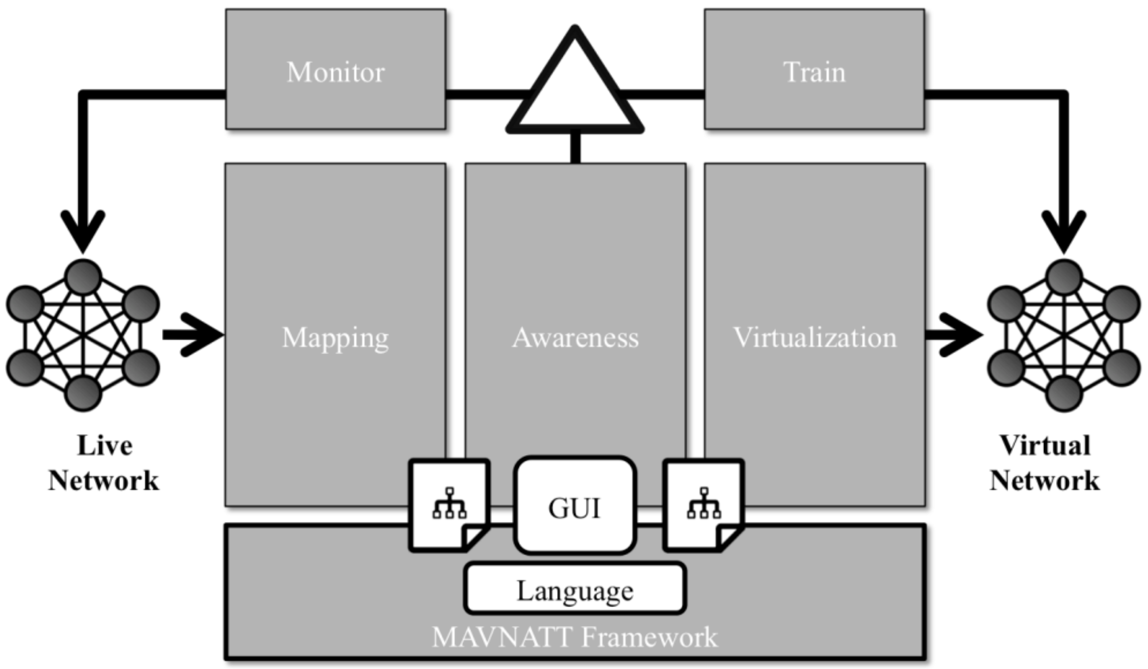


Figure 4. MAVNATT Framework. Source: McBride (2015).

The Awareness Module, which is the focus of this work, is meant to provide awareness of the tactical network with regard to fault management and fault monitoring (McBride, 2015, p. 19). Awareness information is to be displayed to the network administrator via a live graph of the network's physical topology to ensure that real-world faults are immediately recognized and responded to (McBride, 2015, p. 62–63). McBride reviewed SIEMs such as SolarWinds and OpenNMS, which he says can gain awareness of a network, but provide more capability then necessary for MAVNATT. McBride also notes that the reviewed SIEMs require more overhead than desired for MAVNATT, however, he concluded that "the foundation of these solutions and the open-source code can provide a direction that the MAVNATT awareness tool can utilize in its design" (2015, p. 23).

Karaarslan (2017) defined an organizational model called the Cyber Situational Awareness Pyramid (CSAP) to enable CSA for MAVNATT. Figure 5 depicts the CSAP model, which shows that CSA increases as each level of awareness is achieved. Level 1 is Configurational Awareness, where the network administrator acquires or maintains the awareness of the network's adherence to proper configuration. Level 2 is Operational Awareness, in which properly configured cyber security tools are used to detect and eliminate or mitigate threats to the network. Level 3 is Special Conditions Awareness, which adds to the network administrator's CSA by including the network user's awareness of the network's behavior (Karaarslan, 2017, p. 33–35).

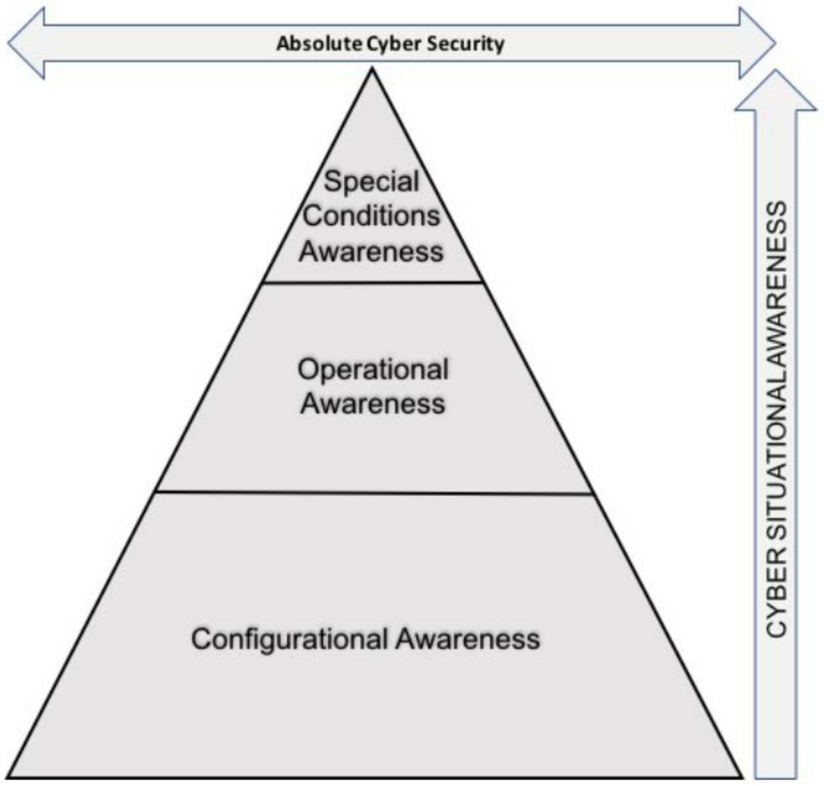

Absolute Cyber Security
Special Conditions Awareness
Operational Awareness
Configurational Awareness
CYBER SITUATIONAL AWARENESS

Figure 5. Cyber Situational Awareness Pyramid. Source: Karaarslan (2017).

A SIEM can effectively provide the information necessary to enable a network administrator to achieve CSA for the first two levels of the CSAP model as those two levels encompass Gaicobe's fixed observational data sources for cyber situational assessment.

D. SUMMARY

This chapter provided an overview of SA and situational assessment, and discussed how CSA extends from SA, and what is necessary to accomplish cyber situational assessment. It also provided a brief overview of the principles of SIEMs and presented three examples of operational SIEM systems. It further provided an overview of MAVNATT, in particular, the Awareness Module, and Karaarslan’s CSAP model. The next chapter discusses a theoretical MAVNATT system design and an Awareness Module design.

THIS PAGE INTENTIONALLY LEFT BLANK

III. AWARENESS MODULE DESIGN

A. APPLICATION DESIGN GOALS

This chapter describes the design objectives of the MAVNATT Awareness Module and provides detail of its subsystems, as well as discussions of the reasoning behind major design choices. From the original vision for MAVNATT, the first objective for this module was to “enhance a network administrator’s situational awareness by providing a visualization platform to display the network topology and status of network devices”; the second objective was to “provide a capability to overlay virtualized devices to support network administrator training and evaluation” (McBride, 2015, p. 60).

The first objective requires that cyber situational assessment be provided to network administrators in order to facilitate their Cyber Situational Awareness (CSA). To do so, it is first necessary to capture and store the baseline details of the network, to include hosts, servers, and other equipment. Once stored, these details can be recalled as necessary for virtualization. Ideally, every detail about the network would be captured for virtualization to provide the greatest fidelity possible in training. This level of detail is necessary to simulate the most complex of scenarios, such as the observation of malware spreading through a network via email injection leading to the initial compromise of a client host, and ultimately a critical server.

The second objective’s requirements parallel that of the first objective. In order to provide a visual overlay of virtualized devices, it is first necessary to obtain cyber situational assessment of the virtualized network through the collection and storage of the details of the virtualized devices. The details of the devices from both the physical and virtual networks can then be recalled and displayed to network administrators and trainers to provide the current cyber situational assessment of both networks. Comparison of these assessments of the physical and virtualized networks to the previously stored baseline details of the physical network will provide a way to compute the deviation of each from the baseline configuration, which will further enhance the cyber situational assessment information that can be presented to network administrators and trainers. The work conducted in this thesis provides the overall plan for MAVNATT to fully meet both of the CSA objectives, as well as supply the Awareness Module with the ability to build the cyber situational assessment information from both the physical and virtual networks.

1. MAVNATT System Design

Through the course of this work, it became necessary to design a framework to support MAVNATT's Awareness Module. The following section discusses a theoretical MAVNATT system design that takes into account guidance from previous work and experience, while drawing on an existing system as a model for MAVNATT to reference. System components, processes, and information flow are discussed, as well as potential system requirements.

a. Guidance and Design Considerations

The mechanical process of obtaining cyber situational assessment from a network can require sampling a very wide variety of sources. MAVNATT must be able to handle any source quickly and easily to be useful. During the design process of the Awareness Module, it was useful to theorize a potential system construct that would fit MAVNATT's original intent, while supporting the module's design objectives. This was used to provide a check on the practicality of the system. This thesis assumes that MAVNATT will be used on tactical networks per the original system concept, which stated that the system should be "simple, lightweight, responsive, and resilient so it can operate across the spectrum of operations and in highly dynamic environments, specifically targeting the portion of the operational network under the oversight and operation of those being trained or evaluated" (McBride, 2015, p. 2).

The author of this thesis has four years' experience on afloat naval platforms and has become familiar with the environmental constraints of tactical networks during that time. That experience suggests that MAVNATT's system design must consist of a single portable server containing the functionality of all three modules, as well as all of the required supporting software and hardware. A single server construct is optimal because the space and resources required to host additional equipment beyond the unit's original requirements is assumed to be prohibitive on many tactical platforms.

To stay within the single server construct, MAVNATT must not be expected to host the virtualization space due to its portability requirement. This space could alternatively be provided by temporary virtualization resources or by the tactical unit's inherent storage infrastructure. One system that has been deployed with this capability is the Navy's Consolidated Afloat Networks and Enterprises Services (CANES) system, which employs virtualization and cloud technologies (Anderson, 2014). For all but very small networks, full-scale network virtualization is not required nor efficient for effective network training. It is reasonable to expect that effective training will only require virtualization of the more crucial aspects of the network and it is possible that tactical units will have sufficient excess resources to accommodate such training.

b. Design Example: OSSIM

Of the SIEMs reviewed in Chapter II, OSSIM is a good example of what the MAVNATT server could be like. The OSSIM platform can be deployed as a single server and uses Debian Linux (Jessie 8) as its operating system. It contains many of the functions needed to produce cyber situational assessment, such as network discovery, vulnerability scanning, and monitoring. Network discovery has yet to be built into the Mapping Module, and using OSSIM as a guide has the potential to save a great deal of research effort. Neither OSSIM nor the Mapping Module has a way to automatically detect and document a network topology at this time, but OSSIM does conduct automated endpoint detection.

OSSIM employs a web-based user interface (WUI) which provides access to nearly all of its functionality. Much of the user accessible information that OSSIM contains, including its WUI configuration and the details of the devices it has been tasked to monitor, is stored within OSSIM's MySQL database schema. This design choice allows OSSIM to maintain the fastest possible access to WUI elements and network details with high referential integrity because the database is maintained in the server's memory. This aspect of OSSIM's design serves as an excellent example for future MAVNATT development.

It is important to remember that OSSIM is the open source offering of AlienVault's USM, which is a commercial product and can change from version to version. While OSSIM offers a great deal of the features that MAVNATT needs, including a functional

WUI that could be adapted to host MAVNATT, it is not a reliable platform on which to build MAVNATT. While researching this work, OSSIM's file system and schema were not openly published and should not be relied upon to remain stable. Any changes AlienVault chooses to make to OSSIM's file system or schema will take place without warning and an update to either could cause MAVNATT's WUI to malfunction or completely break.

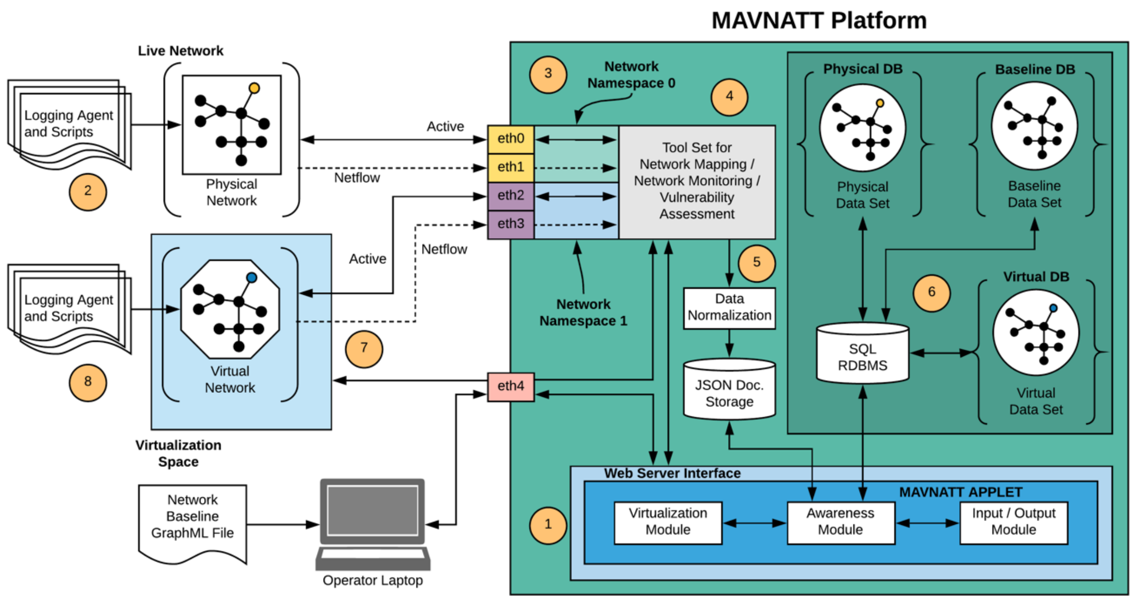


Figure 6. Theoretical MAVNATT System Design.

c. *MAVNATT System: Cyber Situational Assessment Initialization Process Overview*

To create cyber situational assessment, MAVNATT must gather information about the physical network to be virtualized. MAVNATT will accomplish this by utilizing a collection of tools, including the functions of the Mapping Module. All of these tools (Figure 6, block 4) will be accessible to the user through a WUI. The WUI (Figure 6, block 1) will require a webserver and may use a Java servlet container to provide enhanced controls to the user. The webserver would also allow access to the Virtualization and Awareness Modules, along with an Input/Output Module. The purpose of the Input/Output

Module is to parse, convert, and transfer the network baseline information of a Graph Markup Language (GraphML) file into and out of the MAVNATT system.

To capture the network configuration information, scripts and logging agents (Figure 6, block 2) can be deployed to each network component to forward the information as required. This information will be directed to the active logging port on MAVNATT. This port, as well as the passive listening port, will be attached to a network namespace container (Figure 6, block 3). This ensures that network collisions do not occur when MAVNATT is also connected to a virtual network that is logically equivalent to the physical network. File descriptors can be used to connect to network namespaces (Biederman, 2013). Many tools can interface with file descriptors to collect information, but it is likely that some will not have that capability. It may be necessary to install tools that cannot use file descriptors into a software container that natively uses file descriptors to interface with the network stack. Such containers, like the Docker solution used by SEIMonster, create "a container in the containers own network namespace" and would solve this problem (Sriraman, 2017, para. 2).

The network configuration information now moves from the collection tool to a data normalization process (Figure 6, block 5) that will parse through and reformat the information received according to a set of rules to convert it into a uniform format. The output will be pushed to a non-schema database for storage. The Awareness Module can then query this database using an IP address that correlates to a network component to obtain its baseline information. When the Awareness Module has collected all of the required information related to an IP address, that information, along with any network topology data, will be written to the Physical Data Set database within a relational database management system (RDBMS) (Figure 6, block 6). Once the information in the Physical Data Set is complete, that data set will be copied to a second database in the RDBMS to serve as the Baseline Data Set. The Awareness Module will then call the Virtualization Module with the required Baseline data set information to create a virtualized network (Figure 6, block 7). After the virtualization process is complete, scripts and logging agents will be installed to the virtualized network, which will connect to MAVNATT via a separate network namespace (Figure 6, block 3), and then begin pushing virtualized network configuration information into the system.

d. RDBMS Overview

While only one copy of the detailed network configuration is required to determine that a change has been made to the system, the RDBMS (Figure 6, block 6) contains three separate schemas to allow the Awareness Module to track the differences between:

- the Physical Database and the Baseline Database;
- the Virtual Database and the Baseline Database;
- the current state of the physical network from the Physical Database; and
- the current state of the virtual network from the Virtual Database.

To accomplish this, the three databases that the RDBMS contains are:

- the Physical Database, which contains the physical network configuration baseline;
- the Baseline Database, which contains the network configuration baseline; and
- the Virtual Database, which contains the virtualized network configuration baseline.

All three schemas should accommodate the network configuration details from the Physical Database so they may be copied to the Baseline Database and then to the Virtual Database as required. Each schema may contain some tables or fields that are specific to itself, but the tables and fields for all tracked details should be identical for ease of comparison. An indirect comparison of fields may be required if the physical and virtual details are different, which would require an equivalency matrix for the proper comparison of the details. The information that needs to be tracked includes such details as the device’s network identifier, host name, operating system, the amount of RAM installed, and the speed and number of processors onboard. Additionally, the detailed software configuration must be stored, such as the operating system version and any installed updates for the applications on the tracked device. The schema must also store the network topological information.

e. MAVNATT Hardware Requirements

The hardware requirements for the MAVNATT server will generally depend on the number of tools running and the system overhead involved, but they must include a minimum of five physical Ethernet ports to achieve simultaneous monitoring of both the physical and virtual networks, and provide user access to the server. Three port groupings would be created (Figure 6, block 3):

- Two ports dedicated to Physical Network monitoring (eth0 & eth1)
- Two ports dedicated to Virtual Network monitoring (eth2 & eth3)
- One port dedicated to external system access (eth4)

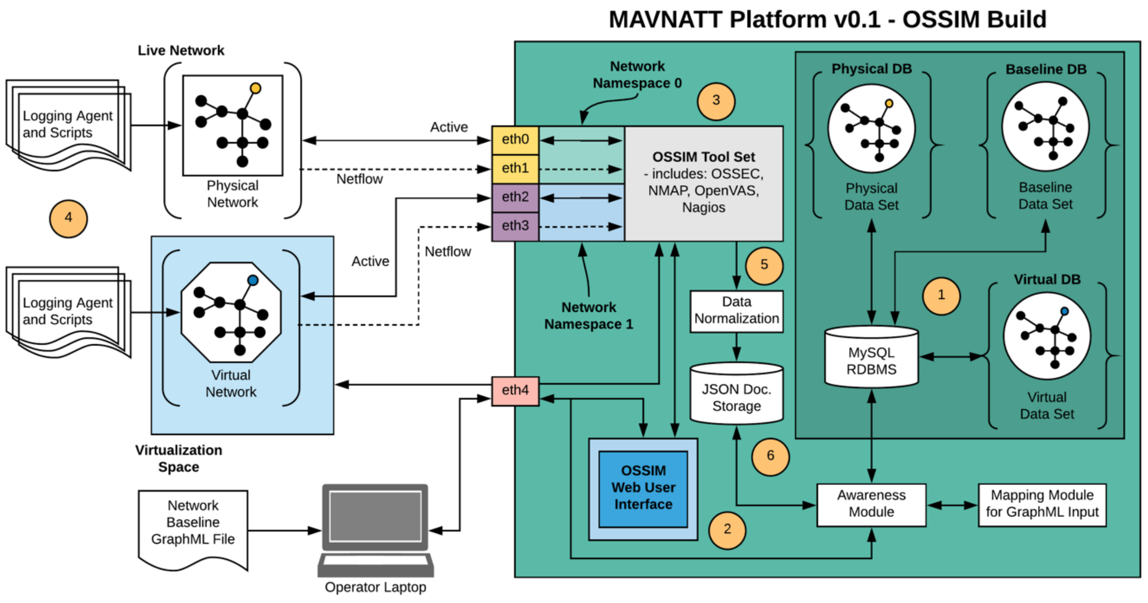


Figure 7. MAVNATT Incremental Design—OSSIM Build.

2. MAVNATT Incremental System Build

It was not possible to implement the proposed system above as part of this thesis due to the extensive amount of development required to achieve it. In support of the future development of MAVNATT’s Awareness Module, the critical database schema was built and its implementation is detailed in Chapter IV to show that creating CSA for MAVNATT is possible. Since an initial system build was not possible, OSSIM was used as an intermediate platform to support schema development.

There were several advantages to using OSSIM as the intermediate platform. The first is that OSSIM can approximate the computational loading of the potential MAVNATT system build, making it a convenient platform for testing. This is important as it is unknown if all of the proposed software components for MAVNATT can run simultaneously on a single machine. OSSIM uses many of the same software components MAVNATT could be expected to use. These components include MySQL (Figure 7, block 1), Apache web server (Figure 7, block 2), and Debian Linux operating system. OSSIM’s tool set (Figure 7, block 3), which includes NMAP for network discovery, OpenVAS for vulnerability detection, and Nagios for network monitoring, could all be incorporated into MAVNATT’s tool set to provide cyber situational assessment. OSSIM has no log storage solution, but it does have the RabbitMQ message broker preinstalled, which would support a logging solution. By adding logging software to OSSIM, it is possible to observe whether a single server could support the MAVNATT system as designed.

An additional advantage is that using OSSIM’s preinstalled software suite should greatly reduce the incremental build time for testing purposes. One such software package that may provide log data to the Awareness Module without a large investment of time is OSSEC, a host-based intrusion detection system that is often used as a log analysis tool and can be used for log normalization. OSSEC includes a host-based agent for the Windows operating system that forwards generated logs to a server running the OSSEC service. Linux/Unix systems already have this capability built-in. Logs are often sent as plain text files and use a format similar to syslog (OSSEC, n.d.)

To collect the host operating system details, a script or application specific to the host operating system can be used to poll and collect the required information and write it to a log file (Figure 7, block 4). This log can then be passed to the OSSEC agent or syslog process for transmission to the OSSEC server on OSSIM for processing. Upon receipt of a log, the activity is checked against a ruleset and then processed accordingly (OSSEC, n.d.). The ruleset is a custom decoder; an XML file that can be written to recognize and parse specific log data. The *ossec-analysisd* process conducts analysis of a log according to the decoder file, which becomes part of the data normalization process (Figure 7, block 5). As noted in Chapter II, Logstash conducts data normalization as well and is an additional candidate to fill this role. Once normalized, the data would be pushed to a log storage solution and be made available to the Awareness Module (Figure 7, block 6). Once the required information was obtained from the log storage solution and further refined, the Awareness Module would then populate the relational database with the newly acquired network configuration details.

Another installed software package within OSSIM that is of immediate use to the work of this thesis is MySQL version 6.3 (Figure 7, block 1), a relational database system. OSSIM's MySQL database requires modification in order to support the storage of all of the necessary configuration details of the network. The details that OSSIM tracks are only a small subset of the information required to accurately virtualize a physical device. This subset is really just the device's network identifiers, host name, and operating system. While the details that OSSIM gathers are useful, it is not worth the effort to reutilize them. To take advantage of them, one would have to know exactly where they are stored with OSSIM's database to query them. OSSIM's database is complex and subject to change upon the release of a new version. Using Oracle's SQL Developer, it was discovered that the database contains 11 schemas with a total of 347 tables. Several of these schemas are used to provide setup information and store SQL functions or procedures. These schemas are heavily interconnected, which means that querying the data may require several complex table JOIN operations in order to obtain the correct data.

It is possible to append the tables currently within the database to store the virtualization and topological data, but as previously discussed, this is risky because those tables may change in later releases of OSSIM, which would cause stability issues for OSSIM as well as MAVNATT. Due to this potential for problems, new schemas within OSSIM's relational database that are tailored specifically to serve each of the three data sets independently were constructed. For this work, having dedicated schemas will allow the data sets to reference OSSIM's schema or even update it if later desired. This would allow for the device network identifier, host name, and operating system details within OSSIM's schema to match the information within the Awareness Module.

The latest version of MySQL, version 8, supports both relational and non-relational database architectures, which gives the potential to optimize the MAVNATT system design by negating the need for two separate database packages (Oracle, 2019). As designed, MAVNATT currently requires relational and non-relational databases. The first is required to maintain normalized, schema-based information, which is accessed via structured query language (SQL), while the second is necessary to store logged information in documents that support a schema-less architecture. Jatana, Puri, Ahuja, Kathuria, & Gosain (2012) note that the advantage of using a non-relational database system to store logs is that it has higher data throughput than relational database systems, allowing it to keep up with incoming logging information more easily.

Finally, OSSIM uses Debian Linux version 8, codenamed Jessie, which carries several advantages of its own. First, its stability and reliability surpass most Linux builds due to its comprehensive package interaction policy, making it very dependable. Debian is thought to support at least 40,000 different software packages (Byfield, 2017), which means there is a good chance MAVNATT can have the functionality it requires at zero cost. Debian itself is also free to use, further keeping development cost to a minimum.

3. MAVNATT Awareness Module Design

The following sub-sections discuss in detail the different components, interfaces, and processes of the proposed MAVNATT Awareness Module Design, as referred to in Figure 8.

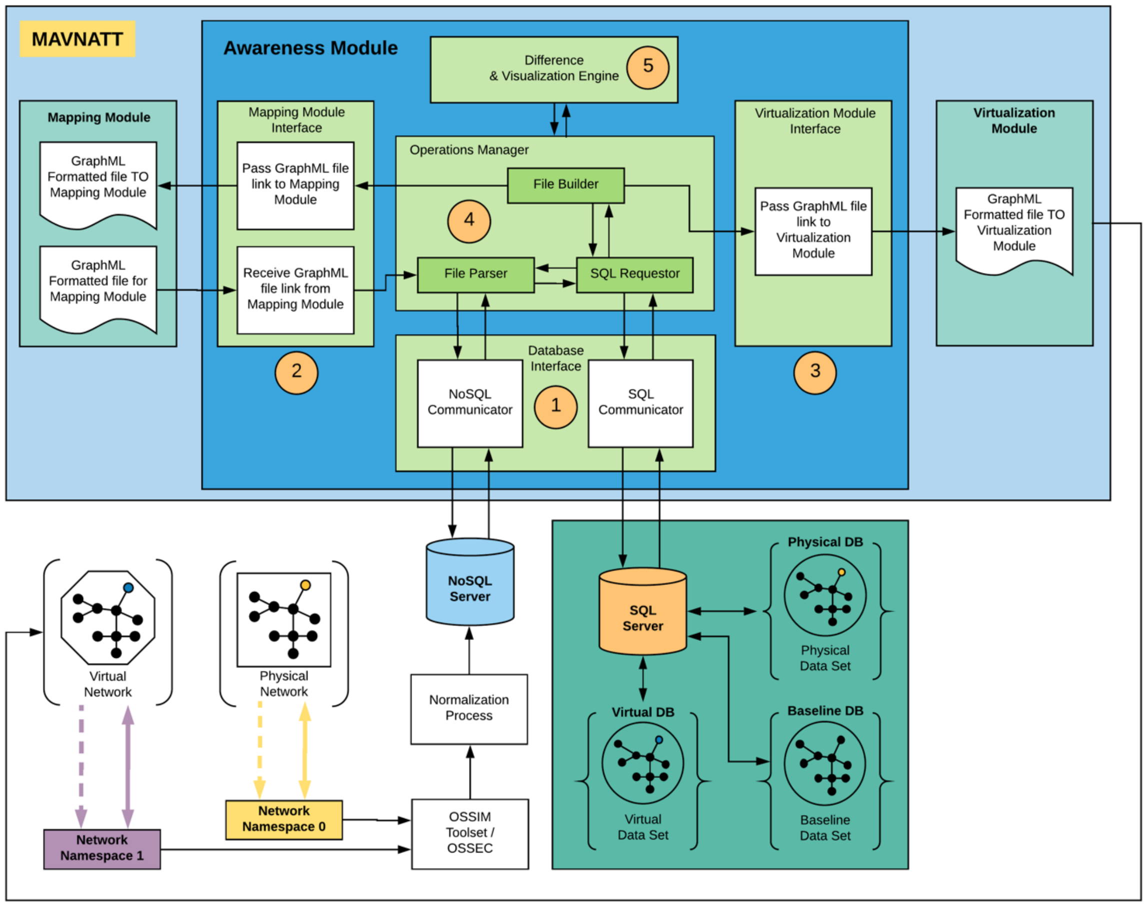


Figure 8. Awareness Module Design.

a. *Database Interface*

The Database Interface (Figure 8, block 1) provides communications between the Awareness Module and the databases within the MAVNATT system. The interface can accept a string that contains a command to transmit and a destination for the transmission. It will then send the command to the proper communication handler to fulfill the request. The interface will accept the returned result of the command from the database into memory and then return that value to the calling function. It will not perform any error checking of the command to be sent, or of the data that is returned.

b. *Mapping Module Interface*

The Mapping Module Interface (Figure 8, block 2) passes GraphML files between the Mapping and Awareness Modules. When called to accept a GraphML file to send to the Awareness Module Operations Manager, the interface will accept a string that is expected to be a GraphML file path. If the file path is valid and the file is of the expected type, it will be passed to the Operations Manager for parsing and inclusion in the proper data set within the relational database. When called to send a file to the Mapping Module from the Operations Manager, the interface will accept a string that is expected to be a GraphML file path. If the file path is valid and the file is of the expected type, it will then be passed to the Mapping Module in order to update and refresh the Mapping Module's visual network map.

c. Virtualization Module Interface

The Virtualization Module Interface (Figure 8, block 3) passes GraphML files from the Awareness Module to the Virtualization Module. Like the Mapping Module Interface, this interface will accept a string that is expected to be a GraphML file path. If the file path is valid and the file is of the expected type, it will then be passed to the Virtualization Module to facilitate the virtualization process.

d. Operations Manager

The purpose of the Awareness Module Operations Manager (Figure 8, block 4) is to oversee the parsing of JSON and GraphML documents, and the compilation of GraphML files, and to manage the Baseline Data Set database within the relational databases. In addition to its own functions, the Operations Manager contains three methods to assist in formatting and moving data between the other modules and the databases:

(1) File Parser

This method is responsible for parsing incoming JSON and GraphML documents to the Operations Manager. Any document that is parsed is expected to have its information sent to the relational database for storage, including all of its visualization details. To accomplish this, the File Parser selects and converts the pertinent information in the incoming GraphML file into a list that will be returned to the Operations Manager.

(2) SQL Requestor

This method is responsible for formatting data into SQL commands for the Operations Manager. It accepts a list that contains a destination within the relational database, a command, and values that it formats into a proper SQL command. It then sends the command to the proper data set via the Database Interface. The result received from this operation is returned to the Operations Manager.

(3) File Builder

This method is responsible for building GraphML files upon receipt of a data source and a list of primary keys. The File Builder will then iterate through primary keys, sending a QUERY command to the SQL Requestor for the information related to the primary key. Upon receipt of a successful request from the SQL Requestor, it will format the data received and append to the GraphML file. When the process is complete, the final file is returned to the Operations Manager.

In addition, the following are activities that the Operations Manager must perform:

(1) Process Incoming GraphML File

Any GraphML file that the Operations Manager receives from the Mapping Module is meant to be inserted into the relational database. Prior to insertion, it will ensure that the user wants to proceed, as this action will permanently modify the database before proceeding with parsing. As each device in the GraphML file is parsed and returned to the Operations Manager, it will immediately be sent to the SQL Requestor for entry into the Baseline Data Set. In the event that the SQL Requestor replies that the entry already exists, the data will be resent as an update to the database. The Operations Manager will inform the user that the operation has completed when the GraphML file is completely parsed.

(2) Purge Database

The user can request to purge the relational database of all device entries through the Operations Manager, which in turn will send SELECT and DELETE commands to the SQL Requestor. The SQL Requestor will return the number of entries that were successfully deleted to the Operations Manager; this information will also be displayed to the user.

(3) Create Network Baseline

When the network administrator believes that enough details have been collected about the network, he can direct the Operations Manager to create the Network Baseline Data Set. The Operations Manager will accomplish this task with several SQL commands, all of which will go through the SQL Requestor and Database Interface.

First, if a Network Baseline Data Set does not already exist, the Operations Manager will create a new database within the relational server capable of storing all of the details necessary to create the Network Baseline Data Set. After the Data Set is created or verified, the Operations Manager will QUERY the relational database to get a full list of primary keys of all of the equipment within the Physical Data Set. Upon receiving the returned list of values, the Operations Manager will iterate through each primary key and transmit an SQL command to copy all of the details associated with that primary key to the new database. Upon completion, the Operations Manager will return the number of primary keys returned from the Physical Data Set and the number of devices written to the Network Baseline Data Set.

(4) Create GraphML File

To create a specific GraphML file using one of the three previously discussed data sets, the Operations Manager issues a SQL command via the SQL Requestor to QUERY the relevant OSSIM MySQL database for a full list of primary keys of all of the equipment within that data set. If the SQL command completes successfully, the Operations Manager then sends it the returned list of primary keys and the source database location to the File Builder, which returns a GraphML file. Upon receiving the GraphML file, the Operations Manager then returns that file to the calling process.

(5) Create Virtual Network Data Set

The creation of the Virtual Network Data Set follows the virtualization of the physical network, which is accomplished by exporting the Network Baseline Data Set to the Virtualization Module via a GraphML file. Once the Virtualization Module has completed the virtualization process, the Operations Manager will then populate the relational database with the information from the Network Baseline Data Set in a way that is analogous to the method used to create the Physical Network Data Set. This will assist in verifying the virtualization process as well as initialize cyber situational assessment for the virtualized network.

e. Difference and Visualization Engine

The purpose of the Difference and Visualization Engine (Figure 8, block 5) is to display the differences between the Physical Network Data Set, the Network Baseline Data Set, and the Virtual Network Data Set by calculating the deviation of the Physical and Virtual Networks from the Network Baseline. The Engine is expected to be controlled by the user and request information from the Operations Manager when a timer event takes place, which could be set to occur once or to repeat many times. It is also expected that the Engine work through the Operations Manager to access the data sets within the relational database. The Engine could do this in two ways.

The first method is by requesting an XML file of an entire data set, which could result in a large file depending on the size of the network and the amount of details required for it. A simpler method would be to request the details for a single network node across all three data sets, compute the differences from the baseline, and then store the result. The visualization of the network could then be updated after the result is stored, depending on the method used to present the visualization. Storage of calculated results within the relational database would enable fast updating of the visualization since the RDBMS is maintained in active memory.

Although the Difference and Visualization Engine is an internal component of the Awareness Module, its design was beyond the scope of this work. During the design process of the Awareness Module, it was useful to theorize how the Engine might work in order to develop the requirements for the Engine within the Awareness Module. This is left for future work on the full implementation of the MAVNATT Awareness Module.

f. *Awareness Visualization Scheme*

Earlier Mapping Module work used a node-link diagram with icons to display a representation of the network map. Color changes on links and nodes were used to represent differences in the network topology baseline from a network scan conducted after the baseline was taken (Collier, 2016, p. 29). It is recommended that the visualization scheme be enhanced to enable it to display the highly detailed information that MAVNATT is capable of attaining due to the Awareness Module designed in this work. The tiered network display in the earlier work is good for visual verification of a network topology, but can make it difficult to view the network in its entirety when it scales to very large size. To mitigate this, the visualization scheme should also include a node-link diagram capable of condensing the network view by reducing the network nodes to points vice icons and using a radial or star display of the edges around nodes with multiple links.

As changes take place on the network, network nodes can change from points back to icons and then be colored according to the Mapping Module's original visualization scheme to bring attention to the change. However, changing colors alone cannot convey the degree to which a change has taken place. For example, changes to a node's software configuration is not accounted for within the Mapping Module's original scheme. A percentage of the amount of details within the software configuration that have deviated from the configuration baseline can be derived from the information within the Awareness Module. This result can then be used to vary the size the network node's icon on the display to show the intensity of the change in the node.

The visualization scheme should also provide the ability to display any data set within the Awareness Module separately or layer data sets together to display a single map. Since there are three data sets (the physical, baseline, and virtual data sets), each could be assigned a default primary color (blue, yellow, or red). The addition of a data set to the display would change the color output of the map. For example, if all three data sets were displayed, the resulting color of the map would be white. If just the Physical and Network Data Sets were displayed, the color of the map would be green. Any network map display used should be interactive and display the details of the node that are known (or unknown) to the system by allowing the user to click on any available node for more information.

g. Proposed Scheme of Numerical Difference Calculation

Displaying the total difference from one data set to another within the network map can be difficult to visualize. As discussed in the previous chapter, there are many aspects of cyber situational assessment that can be quantified. The aspects chosen are often dependent upon the network deployment. To provide a gauge as to how much a particular aspect of the network has changed, deviations can be measured by computing the percentage of change from the original value. For example, a network administrator may want to measure the change in number of devices on the physical network against the baseline network configuration. Any number of desired network aspects could be assigned a percentage value based upon criteria or preference. When added together, their total results would provide a single numerical representation of the amount of deviation in a data set from the baseline configuration.

B. SUMMARY

This chapter has provided a detailed overview of a proposed new construct for the MAVNATT system, and provided an overview of an incremental MAVNATT build using OSSIM as an intermediate platform. It also provided an overview of a proposed MAVNATT Awareness Module, including its functions and internal processes. The next chapter will describe the implementation of the physical, baseline, and virtual schemas, which will provide the Awareness Module with supporting informational framework to enable cyber situational assessment.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MAVNATT RELATIONAL SCHEMA IMPLEMENTATION

A. SCHEMA ATTRIBUTE SELECTION REASONING

The main objective of MAVNATT is to allow network administrators to train on an accurate virtual model of their own network without affecting the network operation. The Awareness Module within MAVNATT is designed to provide an awareness or status of both the physical and virtual networks to enable fault and network difference detection (McBride, 2015 p. 7, 19, 39, 62).

One of the most important aspects of military training is to "train as you fight," meaning that the trainee must train with the same systems and processes they would use in combat. The primary reason for this is to gain system familiarization while learning the required processes and functions of that particular system. In the case of computer network training, the aspects of a physical network that should be virtualized, and therefore the aspects that need to be recorded and tracked, are dependent upon the objectives of the training the administrator is undergoing. The more accurate the virtualization of the network, the more accurate the training environment.

To produce such an environment, the virtualized network should be logically and numerically equivalent to the physical network. Logical equivalency can be achieved by ensuring each node of the virtualized network has the same processes and connections as the physical network, to include having the same networking device connections, open network ports, software processes connected to those open ports, media access control (MAC) addresses, Internet protocol (IP) addresses, and so on. Numerical equivalency is the amount of a particular resource that has been allocated to a physical node as well as to the virtual representation of that node. Numerical equivalency should be considered less important than logical equivalency since resource allocation determines how much of a load a node can handle. For example, a large email server may need a great deal of processing, memory, and storage space to function efficiently. However, that server may not need as many resources when virtualized to accomplish a network training objective. Quantities of attributes such as CPU cores, allocated memory, allocated video memory, etc., will be collected for each node and then be referenced to create a virtualized node that can properly accommodate the training objectives to be achieved.

The minimum set of attributes that MAVNATT must keep track of are those maintained in the GraphML file used between the Mapping and Virtualization Modules. These attributes include an image of the node, which may be all that is needed to meet the required training objective. In the event that a GraphML file is unavailable, MAVNATT needs to compile the logical and numerical system attributes that contribute the most to meeting network administrator training objectives efficiently with regard to virtualization time and resources.

The initial attribute list created for this work was based on those attributes considered to the have the greatest impact in creating a virtual representation of a physical network with enough accuracy to meet a majority of the training objectives for network administrators. Logical and numerical attributes that VirtualBox can implement or adjust that provided this impact were identified and included in the schema described below. Having these attributes in advance of the virtualization process will also reduce the time to implement that process since VirtualBox's API allows them to be done programmatically vice implementing them manually. A few additional attributes that have little impact on training objectives, such as device manufacturer and device model, were also added to assist with troubleshooting.

B. SCHEMA ATTRIBUTE MATRIX

The Attribute Matrix (Appendix A) contains a list and description of the 65 selected logical, numerical, and informational attributes deemed most useful toward building a faithful representation of the network for the system admin training. A description of the columns follows:

Attribute Matrix: Column Name and Descriptions

A# - Attribute Number: provides an index for table and field references in the schema description.

Employed Upon/After Creation: Denotes if the attribute can be included into the VM upon or after its creation.

Expected Training Impact: Assigns an expected training impact to the virtualization attribute.

- High – Will have a serious impact on the accuracy of the training because it is a logical attribute of the physical network.
- Medium – May have a noticeable impact on the smoothness of the training depending upon the amount of resources (numerical attribute) required to replicate the services within the virtual network environment.
- Low – Any details in this category are mostly for organization and management of the virtualization and could have an impact on the training if left out.
- None – These are details which have minimal to no effect on the accuracy of the virtualization because they are physical node metadata or data points which cannot be virtualized at this time.

Manual Section / Page#: Section and page number within the VirtualBox Manual where the attribute was found.

Attribute Type: Denotes the attribute type.

- Logical Attribute – A discrete value or specific setting for a component of a node.
- Numerical Attribute – A quantitative value that describes the resource allocation for a component of a node.
- Informational Attribute – Metadata for a component of a node.

Component Category: Denotes if the detail belongs to a node's hardware or software. This is an important aspect of virtualization, as highly accurate hardware emulation can be difficult to achieve (Kuchera, 2011). In the event that it is necessary to virtualize specialized equipment, the difference in accuracy between the physical and virtualized node may be something that could affect the accuracy of training taking place and should be noted by the trainer.

Component Type: Denotes the computer hardware or software category the detail generally relates to.

Component Description: Specific description of the detail.

VirtualBox Command: VirtualBox command used to implement the detail.

VirtualBox Command Attributes: Detail of the attributes for the VirtualBox command, if available. Shaded cells represent information that is expected to not directly translate between the physical and virtual databases due to VirtualBox hardware emulation options.

C. SCHEMA DESIGN CONSIDERATIONS

Three criteria were taken into consideration during the design of the schema. The first is that the designed schema should be implemented as a universal solution for the physical, baseline, and virtual databases to enable direct comparisons between each. To make this possible, the baseline database includes a variation in its schema that allows for a logical transition between all three databases. This variation is the addition of a single field, where necessary, to each of the baseline database's look up tables to track the translation between the attribute on the physical node and the VirtualBox option chosen to emulate that attribute. The identified tables that required this modification are hardware related, yet this could be applied to any look up table where a direct comparison is not possible. To reduce any confusion between direct comparisons of the databases, VirtualBox categories and terminology were used to group and name attributes as much as possible. Most grouping within the schema was based upon the Attribute Matrix's category and component columns, which were derived from the VirtualBox manual and experience.

The second consideration was that the schema should provide for the enforcement of virtualization limits, to the extent practical. VirtualBox was noted to have several limitations in the number of network interface cards, serial ports, and other interfaces that it can support. Additionally, all of the devices still had to have unique settings to prevent conflicts, just as any physical computer would have as well. To ensure that the schema was able to support unique combinations of virtualized equipment with a node, junction and lookup tables were employed to provide referential integrity while enforcing unique constraints on equipment settings where necessary. This also assists in the virtualization of the node since the lookup tables only contain the options that are available to the commands within VirtualBox. As VirtualBox changes, the options within these tables will have to be updated. Physical nodes may not have the same constraints as what VirtualBox can virtualize, so only the virtual database is expected to use prepopulated lookup tables, which will limit how many or what kind of selection can be made for a database entry.

The final consideration was the optimization of the database's memory usage through the reduction of redundant data by reusing entries as many times as possible. This was accomplished through the use of lookup tables and unique constraints on row entries for those tables.

D. MAVNATT SCHEMA DESIGN DIAGRAM

The schema design, based on the logical grouping of attributes with regard to the design considerations, is shown in Appendix B. The schema design is for all three databases, with the extra tables and fields for the baseline database highlighted in blue. The design elements include data and table types as follows:

1. Data Types

The data types selected for each field were dependent upon the amount of data that the field was expected to contain, based on a 1,000-node network. The data limitations can be adjusted higher to allow greater capacity if necessary. The following data types were used:

- TINYINT UNSIGNED: An unsigned integer type that ranges from 0 to 255.
- SMALLINT UNSIGNED: An unsigned integer type that ranges from 0 to 65535.
- MEDUIMINT UNSIGNED: An unsigned integer type that ranges from 0 to 16777215.

- VARCHAR(M): Variable-length string with length M. Most fields in the tables are expected to contain no more than 16 characters. These limits were exercised to make the database more space efficient and the size expectations were derived from the VirtualBox manual and the attributes that can be assigned to commands. String lengths that are not of 16 are documented in the table description as to what it is expected to contain. All string lengths are to be considered initial values that have not been tested and should be changed as required.

2. Table Types

Many of the tables in the schema have a parent-child relationship to create reference tables that can be used by other tables to lookup specific data thru JOIN operations. In the case of the virtual database, the tables that contain prepopulated entries are intended to restrict attribute detail selection options to a specific set while reducing redundant information entries in the database.

In addition to the above lookup tables, junction tables were also used in the schema as a way of assigning a specific configuration set to a device or to the node itself. This was accomplished through the use of two fields within the table that hold foreign keys, each of which referenced the unique primary key of another table. A primary key constraint is then placed on the combination of the two foreign keys to ensure that its entry in the table is unique. This process ensures that the device or configuration cannot be assigned to any other node or device.

3. Schema Redundancy

The node’s IP address is duplicated within the schema in both the topology table and the NIC Configuration tables. This is done to account for the case where the topology does not exist for a node; thus, the IP address must still be recorded in the NIC Configuration table. This is especially important if the distant node is outside of the virtualized network.

E. SCHEMA TABLE AND FIELD DESCRIPTIONS

A detailed description of the schema tables, to include their fields and relationships, is provided in Appendix C. The description seeks to clarify the logical structure of the schema, as well as how it employs unique constraints to enforce virtualization limitations. The table field descriptions include the field's data-type, reasoning for that data-type if necessary, and a brief description of what the field contains. Some of the fields are linked to the Attribute Matrix (Appendix A) through the use of the attribute number (A#), which can be found immediately after the bold text field name. Much of the schema description is written with regard to how it will be employed as the Virtual Database as this is the most restrictive use case. Any difference within the schema's look up tables is noted within the description.

F. MAVNATT SCHEMA SQL CODE

The schema build code and the virtual database lookup table population code is provided in Appendix D. Oracle's SQL Developer v18 was the environment used to build the code, which was tested on OSSIM v5.5 with MySQL v6.3 installed. The resulting database was then analyzed to ensure table connections were accurately generated. This was accomplished via Oracle's SQL Developer Data Modeler function by importing the baseline database's Data Dictionary and viewing the resulting relational table map (Appendix E).

G. SUMMARY

This chapter provided a detailed discussion of the design and implementation of a relational database system used to provide the MAVNATT Awareness Module with a way to create cyber situational assessment of a network via the recording of its logical and numerical attributes. It discussed node attribute selection, design considerations, and the overall SQL development solution. The next chapter assesses how effectively this RDBS system implementation meets the system requirements of MAVNATT and discusses future work to further these research efforts.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE WORK

A. OVERVIEW

This research has defined a conceptual model of a top-down system to virtualize a computer network through the cyber situational assessment of that network for the purpose of training network administrators. To help create a faithful virtualized representation of the network, we propose a collection of logical and numerical attributes that can be used to create the virtualized network. The models and processes discussed within this work expand what can be expected from the original MAVNATT conceptual model. The chapter details the conclusions resulting from the research and then focuses on future research needs to finalize the Awareness Module and a full MAVNATT build.

B. CONCLUSIONS

The research objective for this work was to determine how situational awareness could be generated for MAVNATT. The development of the conceptual designs for MAVNATT's Awareness Module drove the selection of attributes of network nodes, which led to the realization of the need to employ a data structure capable of managing and tracking nodal network attributes in an efficient manner. To this end, a relational schema was used to manage and track the multitude of attributes necessary to enable the generation of cyber situational assessment.

1. Research Objectives

There were two main objectives for this work. The first objective was to find an efficient method to provide the tactical network administrator with Cyber Situational Awareness (CSA). The background research noted that it is only possible to provide CSA through cyber situational assessment, since CSA is a cognitive process. To provide cyber situational assessment, this work instead offers a conceptual system of several tools and their information flows to collect, normalize, and store incoming data. This processing of information actually applies to the Mapping Module and expands what the module should be considered to do. Instead of just mapping the topology of the network, the Mapping

Module should also examine each node to identify and capture the details of its baseline configuration. These details can be captured as a set of attributes that are then organized in a relational database. Once organized, this data becomes available for consumption by the Awareness Module to enable cyber situational assessment.

The second research objective was to implement the module for MAVNATT that provides baseline configurational awareness to the network administrator. This objective was not accomplished because it was discovered that the amount of effort necessary to design and implement a fully functional Awareness Module was far greater than anticipated for a single M.Sc. thesis. However, by providing a conceptual design for the Awareness Module with a supporting MAVNATT system design concept, initial construction and testing of both the MAVNATT system and the Awareness Module can now take place. The designed and implemented relational schema provides a blueprint to provide cyber situational assessment via the management and tracking of the desired network attributes.

2. Research Questions

Primary question: How is the CSA of a tactical network achieved?

Conclusion: The CSA of a tactical network is achieved via the collection, normalization, and recording of network attributes. The software tools required to process the attributes include host agents to collect and send the required information; a logging system to receive and parse the information from the host agents and passive netflow collection; normalization processes for the received attributes; and an indexed storage system by which the attributes can be retrieved. The attributes may require additional processing prior to presentation to the user, such as management and tracking of the attribute, or comparison with a previously recorded state. The finalized data attributes are then presented in a clear and concise way that efficiently relays the available information to the user, creating cyber situational assessment.

Secondary question: What kind of information needs to be collected, and how should it be analyzed and visualized to produce effective CSA?

Conclusion: The kind of information to be collected, how it is analyzed, and how it is displayed all depend on the aspect of the network which the user wishes to produce for effective cyber situational assessment. MAVNATT's goal is tactical network administrator training. To produce cyber situational assessment in support of that goal means gathering the baseline configuration of all of the nodes of the network to be virtualized. This enables the trainees and trainers to see the difference between the physical network and the virtual network when compared to the baseline. This baseline configuration includes each node's hardware specifications, network configuration, operating system configuration, and user settings.

Tactical network administrator training could also require network security aspects as well. This would require collecting the network security attributes, which the implemented schema does not fully support. It is important to emphasize that the relational schema produced by this work can and should be further extended and refined.

C. FUTURE WORK

Since this work focused on the design of the MAVNATT system and the Awareness Module, the implementation and testing of those designs are certainly opportunities for future work. However, there are many other opportunities to refine and improve this work beyond system implementation and testing. There is much research left to be done in order to fully support the training aspect of MAVNATT.

1. Configuration Baseline Research

This work compiled many of the configuration baseline attributes that VirtualBox can virtualize; however, the list of attributes is not final. Most of the compiled attributes are hardware-based and only a few operating system details are captured by our relational schema. A survey of tactical network administrator training goals and the attributes required to meet those goals needs to be conducted in order to validate the configuration baseline attributes that the implemented schema focuses on. This will determine the comprehensive set of attributes that should be collected and tracked with regard to the tactical network administrator training that MAVNATT is meant to support.

2. Cyber Security Situational Assessment

As previously mentioned, cyber security is an aspect of tactical network administrator training. Attributes of cyber security could be incorporated into MAVNATT's relational schema to provide cyber situational assessment of both the physical and virtual networks, but the necessary attributes to manage and track would need to be researched and identified first. This would not be a trivial work as processes running within encrypted spaces may be impossible to access. However, there are a great many indicators that can be used to detect malicious activity. The resulting attributes required to make that detection and create cyber security situational assessment may be very large, which would make the display of those attributes to the user problematic. To convey this aspect of cyber situational assessment efficiently would then require that MAVNATT process and condense individual attributes into a manageable report. This processing and condensing of information will require various algorithms to be developed to pull meaning out of the attributes. Those algorithms could then be applied to machine learning to efficiently create that meaning.

3. Difference Engine Research and Construction

As the number of attributes that MAVNATT's relational schema tracks and manages grows, so too will the difficulty in displaying to the user the differences between the operational and virtual versions of these attributes. An efficient and dynamic difference engine that can incorporate with MAVNATT's Awareness Module will be crucial to ensuring that cyber situational assessment is efficiently conveyed to the user.

APPENDIX A. ATTRIBUTE MATRIX

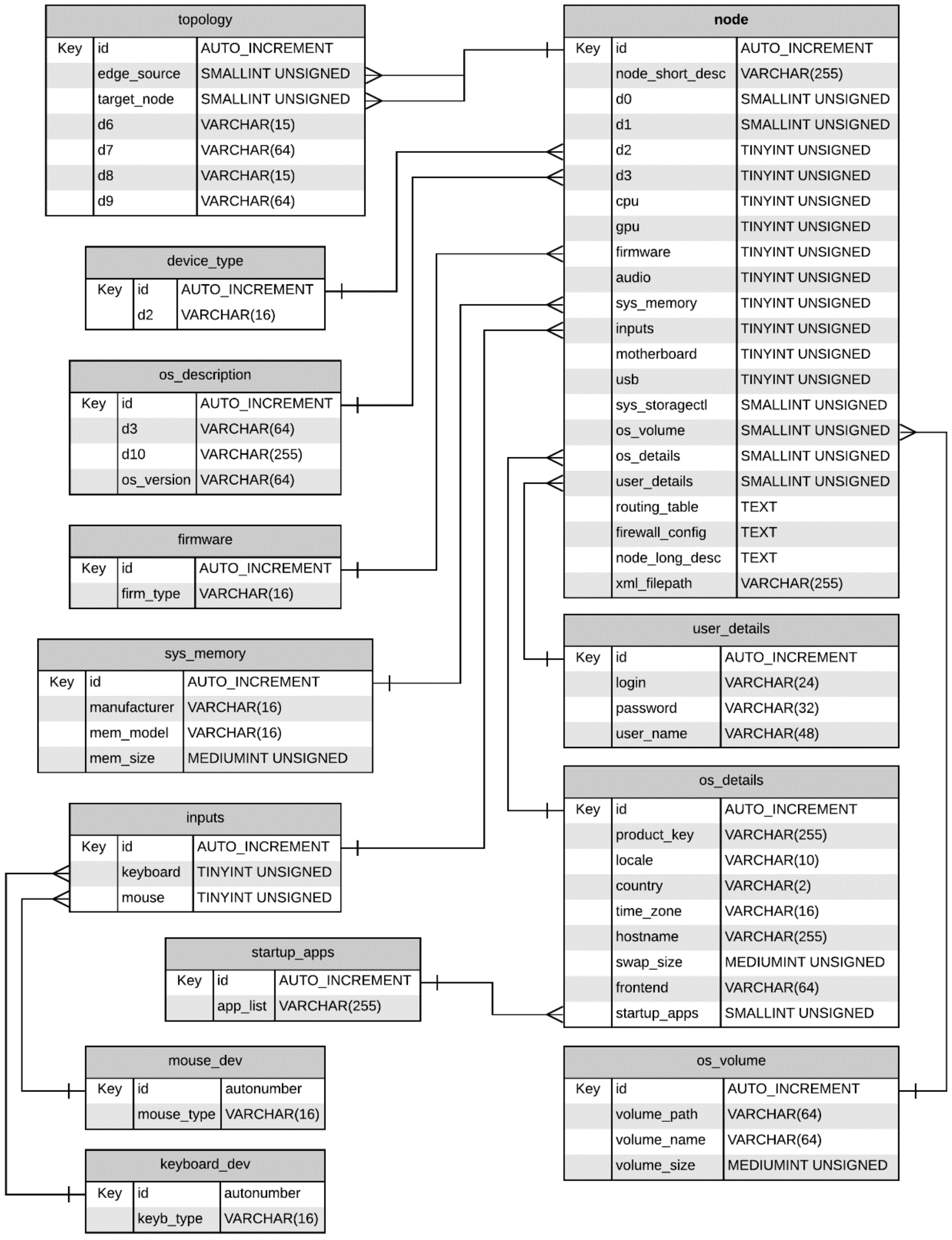
A#	Employed Upon/After Creation	Expected Training Impact	Manual Section/ Page#	Attribute Type	Component Category	Component Type	Component Description	VirtualBox Command	VirtualBox Command Attributes
1	After	Medium	8.8.3/142	Logical	Hardware	Audio	Support Type	modifyvm	--audio none|null|dsound|oss|alsa|pulse|coreaudio
2	After	Medium	8.8.3/142	Logical	Hardware	Audio	Controller	modifyvm	--audiocontroller ac97|hda|sb16
3	After	Medium	8.8.3/142	Logical	Hardware	Audio	Input On/Off	modifyvm	--audioin on
4	After	Medium	8.8.3/142	Logical	Hardware	Audio	Output On/Off	modifyvm	--audioout on
5	After	Medium	8.8.1/136	Numerical	Hardware	CPU	# of cores	modifyvm	--cpus <cpucount>
6	After	High	8.8.1/136	Logical	Hardware	CPU	Type	modifyvm	--cpu-profile <host|intl 80[86|286|386]>
7	Informational	None		Informational	Hardware	CPU	Manufacturer		
8	Informational	None		Informational	Hardware	CPU	Model		
9	After	High	8.8.1/138	Logical	Hardware	Firmware	Type	modifyvm	--firmware bios|efi|efi32|efi64
10	After	High	8.8.1/138	Logical	Hardware	Firmware	Boot Order	modifyvm	--boot <1-4> none|floppy|dvd|disk|net
11	After	Medium	8.8.1/136	Numerical	Hardware	Graphics	VRAM Size	modifyvm	--vram <vramsize>
12	After	Medium	8.8.1/138	Logical	Hardware	Graphics	Controller Type	modifyvm	--graphicscontroller none|vboxvga|vmsvga|vboxsvga
13	After	Low	8.8.3/141	Logical	Hardware	Input Device	Mouse	modifyvm	--mouse <ps2|usb|usbtablet|usbmultitouch>
14	After	Low	8.8.3/141	Logical	Hardware	Input Device	Keyboard	modifyvm	--keyboard <ps2|usb>
15	Informational	None		Informational	Hardware	Memory	Manufacturer		
16	Informational	None		Informational	Hardware	Memory	Type/Model		
17	After	Medium	8.8.1/136	Numerical	Hardware	Memory	Size	modifyvm	--memory <memorysize>
18	Informational	None		Informational	Hardware	Motherboard	Manufacturer		
19	Informational	None		Informational	Hardware	Motherboard	Type/Model		
20	After	High	8.8.1/136	Logical	Hardware	Motherboard	ACPI	modifyvm	--acpi on|off

A#	Employed Upon/After Creation	Expected Training Impact	Manual Section/ Page#	Attribute Type	Component Category	Component Type	Component Description	VirtualBox Command	VirtualBox Command Attributes
21	After	High	8.8.1/136	Logical	Hardware	Motherboard	I/O	modifyvm	--ioapic on|off
22	After	High		Logical	Hardware	Networking	NIC Device Numbers		<1-N> = 1 thru 8
23	After	High	8.8.2/139	Logical	Hardware	Networking	Hardware Type	modifyvm	--nictype<1-N> Am79C970A|Am79C973|82540EM|82543GC|82545EM|virtio
24	After	High	8.8.2/139	Logical	Hardware	Networking	Mode	modifyvm	--nic<1-N> none|null|nat|natnetwork|bridged|intnet|hostonly|generic
25	After	High	8.8.2/139	Logical	Software	Networking	Promiscuity	modifyvm	--nicpromisc<1-N> deny|allow-vms|allow-all
26	After	High	8.8.2/139	Logical	Software	Networking	Boot Priority	modifyvm	--nicbootprio<1-N> <priority>
27	After	High		Logical	Software	Networking	IP Address		
28	After	High		Logical	Software	Networking	DNS		
29	After	High		Logical	Software	Networking	Gateway		
30	After	High	8.8.2/140	Logical	Software	Networking	MAC Address	modifyvm	--macaddress<1-N> auto|<mac>
31	After	High		Logical	Software	Networking	Routing Table		
32	After	High		Logical	Software	Networking	Firewall Config		
33	After	High	8.8.2/140	Logical	Software	Networking	Vbox Internal Networking	modifyvm	--intnet<1-N> network
33	After	High	8.8.2/140	Logical	Software	Networking	NAT Network Name	modifyvm	--nat-network<1-N> <network name>
33	After	High	8.8.2.1 /140	Logical	Software	Networking	NAT IP Range	modifyvm	--natnet<1-N> <network>|default
34	After	High	8.8.2.1/ 103, 140	Logical	Software	Networking	NAT Port Forwarding Rule	modifyvm	--natpf<1-N> [<name>],tcp|udp,[<hostip>],<hostport>,[<guestip>],<guestport>
35	Upon	High	8.44/204	Logical	Software	OS	login	unattended install	--user=<login>
36	Upon	High	8.44/204	Logical	Software	OS	password	unattended install	--password=<password>
37	Upon	Low	8.44/204	Logical	Software	OS	user name	unattended install	--full-user-name=<name>
38	Upon	High	8.44/204	Logical	Software	OS	Product Key	unattended install	--key=***<product-key>***

A#	Employed Upon/After Creation	Expected Training Impact	Manual Section/ Page#	Attribute Type	Component Category	Component Type	Component Description	VirtualBox Command	VirtualBox Command Attributes
39	Upon	High	8.44/204	Logical	Software	OS	Locale	unattended install	--locale=***<ll_CC>***
40	Upon	High	8.44/204	Logical	Software	OS	Country Code	unattended install	--country=***<CC>***
41	Upon	High	8.44/204	Logical	Software	OS	Time Zone	unattended install	--time-zone=***<tz>***
42	Upon	High	8.44/204	Logical	Software	OS	Hostname	unattended install	--hostname=***<fqdn>***
43	After	Medium		Numerical	Software	OS	Swap File Size		
44	After	High	8.13/151	Informational	Software	OS	Frontend	modifyvm	--defaultfrontend default|<name>
45	After	High		Logical	Software	OS	Startup Apps		
46	Informational	High		Logical	Software	OS	Verison		
47	Upon	Low	8.7/135	Informational	Software	OS	VM Name	createvm	name <name>
48	After	Low	8.8/135	Informational	Software	OS	VM Description	modifyvm	--description <desc>
49	After	High		Logical	Software	OS/Storage	Volume Drive Path/Letter		
50	After	Medium		Logical	Software	OS/Storage	Volume Name		
51	After	Low		Numerical	Software	OS/Storage	Volume Size		
52	After	High	8.8.3/141	Logical	Hardware	Ports	Serial Port	modifyvm	--uart<1-N> off|<I/O base> <IRQ>
53	After	High	8.8.3/141	Logical	Hardware	Ports	Serial Mode	modifyvm	--uartmode<1-N> <arg>
54	After	High	8.8.3/142	Logical	Hardware	Ports	LPT Mode	modifyvm	--lptmode<1-N> <Device>
55	After	High	8.8.3/142	Logical	Hardware	Ports	LPT	modifyvm	--lpt<1-N> <I/O base> <IRQ>
56	Informational	None		Informational	Hardware	Storage	Manufacturer		
57	Upon	High	8.20/164	Informational	Hardware	Storage Ctrl	Storage Ctrl Name	storagectl	name <name>
58	Upon	High	8.20/164	logical	Hardware	Storage Ctrl	Storage Type	storagectl	--add ide|sata|scsi|floppy|sas|usb|pcie
59	Upon	High	8.20/164	logical	Hardware	Storage Ctrl	Controller Type	storagectl	--controller LSILogic|LSILogicSAS|BusLogic|IntelAhci|PIIX3|PIIX4|ICH6|I82078|USB|NVMe]

A#	Employed Upon/After Creation	Expected Training Impact	Manual Section/ Page#	Attribute Type	Component Category	Component Type	Component Description	VirtualBox Command	VirtualBox Command Attributes
60	Upon	High	8.20/164	Logical	Hardware	Storage Ctrl	Storage Port Count	storagectl	--portcount <1-30>
61	Upon	High	8.20/164	Logical	Hardware	Storage Ctrl	Storage Host I/ O Cache	storagectl	--hostiocache on|off
62	Upon	High	8.20/164	Logical	Hardware	Storage Ctrl	Bootable Storage Ctrl	storagectl	--bootable on|off
63	After	High	8.8.3/142	Logical	Hardware	USB	Version 1	modifyvm	--usb on|off
64	After	High	8.8.3/142	Logical	Hardware	USB	Version 2	modifyvm	--usbehci on|off
65	After	High	8.8.3/142	Logical	Hardware	USB	Version 3	modifyvm	--usbxhci on|off

APPENDIX B. MAVNATT SCHEMA DESIGN



topology		
Key	id	AUTO_INCREMENT
	edge_source	SMALLINT UNSIGNED
	target_node	SMALLINT UNSIGNED
	d6	VARCHAR(15)
	d7	VARCHAR(64)
	d8	VARCHAR(15)
	d9	VARCHAR(64)

node		
Key	id	AUTO_INCREMENT
	node_short_desc	VARCHAR(255)
	d0	SMALLINT UNSIGNED
	d1	SMALLINT UNSIGNED
	d2	TINYINT UNSIGNED
	d3	TINYINT UNSIGNED
	cpu	TINYINT UNSIGNED
	gpu	TINYINT UNSIGNED
	firmware	TINYINT UNSIGNED
	audio	TINYINT UNSIGNED
	sys_memory	TINYINT UNSIGNED
	inputs	TINYINT UNSIGNED
	motherboard	TINYINT UNSIGNED
	usb	TINYINT UNSIGNED
	sys_storagectl	SMALLINT UNSIGNED
	os_volume	SMALLINT UNSIGNED
	os_details	SMALLINT UNSIGNED
	user_details	SMALLINT UNSIGNED
	routing_table	TEXT
	firewall_config	TEXT
	node_long_desc	TEXT
	xml_filepath	VARCHAR(255)

device_type		
Key	id	AUTO_INCREMENT
	d2	VARCHAR(16)

os_description		
Key	id	AUTO_INCREMENT
	d3	VARCHAR(64)
	d10	VARCHAR(255)
	os_version	VARCHAR(64)

firmware		
Key	id	AUTO_INCREMENT
	firm_type	VARCHAR(16)

sys_memory		
Key	id	AUTO_INCREMENT
	manufacturer	VARCHAR(16)
	mem_model	VARCHAR(16)
	mem_size	MEDIUMINT UNSIGNED

inputs		
Key	id	AUTO_INCREMENT
	keyboard	TINYINT UNSIGNED
	mouse	TINYINT UNSIGNED

startup_apps		
Key	id	AUTO_INCREMENT
	app_list	VARCHAR(255)

mouse_dev		
Key	id	autonumber
	mouse_type	VARCHAR(16)

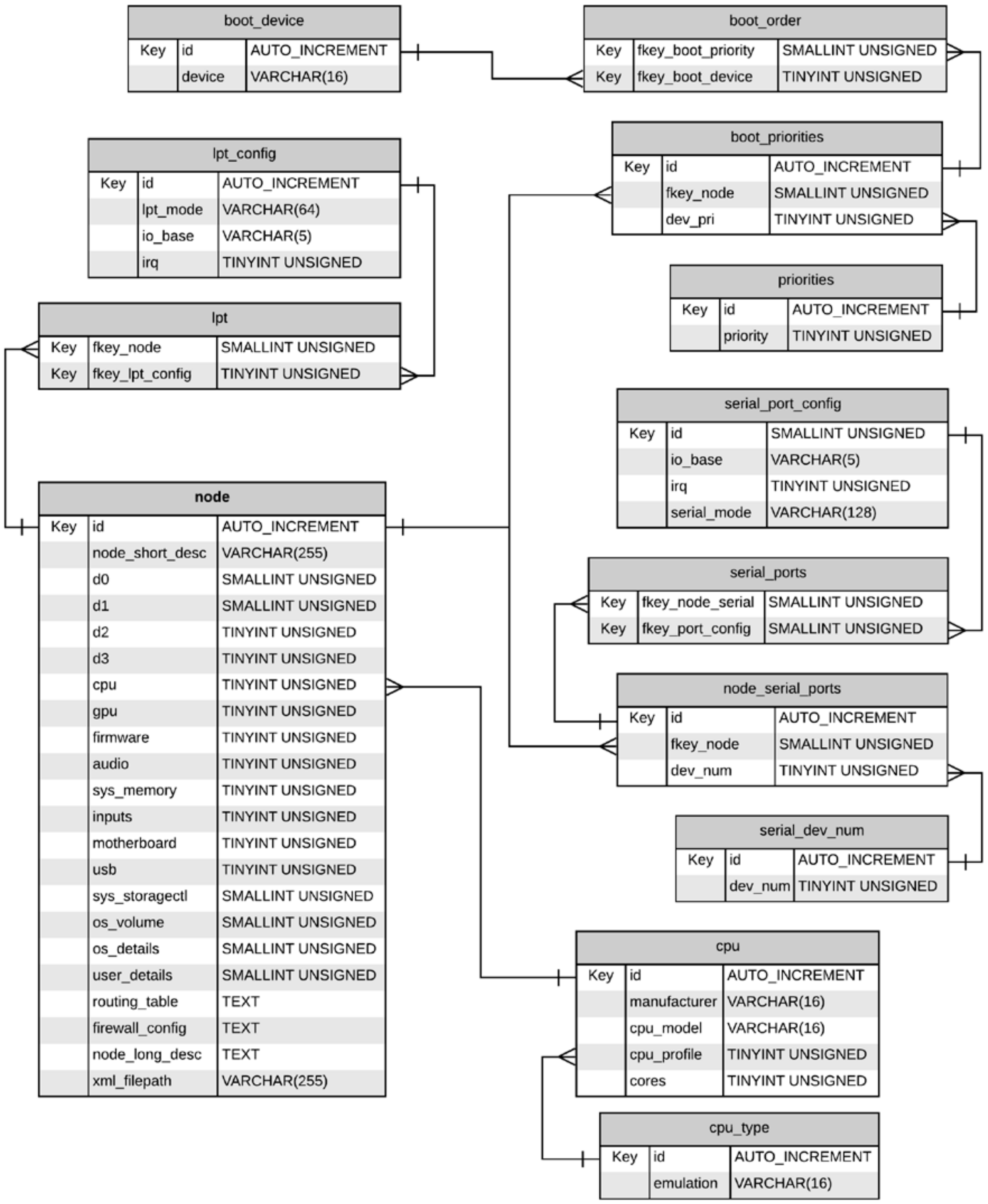
keyboard_dev		
Key	id	autonumber
	keyb_type	VARCHAR(16)

user_details		
Key	id	AUTO_INCREMENT
	login	VARCHAR(24)
	password	VARCHAR(32)
	user_name	VARCHAR(48)

os_details		
Key	id	AUTO_INCREMENT
	product_key	VARCHAR(255)
	locale	VARCHAR(10)
	country	VARCHAR(2)
	time_zone	VARCHAR(16)
	hostname	VARCHAR(255)
	swap_size	MEDIUMINT UNSIGNED
	frontend	VARCHAR(64)
	startup_apps	SMALLINT UNSIGNED

os_volume		
Key	id	AUTO_INCREMENT
	volume_path	VARCHAR(64)
	volume_name	VARCHAR(64)
	volume_size	MEDIUMINT UNSIGNED

Figure 9. Relational Schema Diagram—View 1 of 4.



boot_device		
Key	id	AUTO_INCREMENT
	device	VARCHAR(16)

boot_order		
Key	fkey_boot_priority	SMALLINT UNSIGNED
Key	fkey_boot_device	TINYINT UNSIGNED

lpt_config		
Key	id	AUTO_INCREMENT
	lpt_mode	VARCHAR(64)
	io_base	VARCHAR(5)
	irq	TINYINT UNSIGNED

boot_priorities		
Key	id	AUTO_INCREMENT
	fkey_node	SMALLINT UNSIGNED
	dev_pri	TINYINT UNSIGNED

priorities		
Key	id	AUTO_INCREMENT
	priority	TINYINT UNSIGNED

lpt		
Key	fkey_node	SMALLINT UNSIGNED
Key	fkey_lpt_config	TINYINT UNSIGNED

serial_port_config		
Key	id	SMALLINT UNSIGNED
	io_base	VARCHAR(5)
	irq	TINYINT UNSIGNED
	serial_mode	VARCHAR(128)

node		
Key	id	AUTO_INCREMENT
	node_short_desc	VARCHAR(255)
	d0	SMALLINT UNSIGNED
	d1	SMALLINT UNSIGNED
	d2	TINYINT UNSIGNED
	d3	TINYINT UNSIGNED
	cpu	TINYINT UNSIGNED
	gpu	TINYINT UNSIGNED
	firmware	TINYINT UNSIGNED
	audio	TINYINT UNSIGNED
	sys_memory	TINYINT UNSIGNED
	inputs	TINYINT UNSIGNED
	motherboard	TINYINT UNSIGNED
	usb	TINYINT UNSIGNED
	sys_storagectl	SMALLINT UNSIGNED
	os_volume	SMALLINT UNSIGNED
	os_details	SMALLINT UNSIGNED
	user_details	SMALLINT UNSIGNED
	routing_table	TEXT
	firewall_config	TEXT
	node_long_desc	TEXT
	xml_filepath	VARCHAR(255)

serial_ports		
Key	fkey_node_serial	SMALLINT UNSIGNED
Key	fkey_port_config	SMALLINT UNSIGNED

node_serial_ports		
Key	id	AUTO_INCREMENT
	fkey_node	SMALLINT UNSIGNED
	dev_num	TINYINT UNSIGNED

serial_dev_num		
Key	id	AUTO_INCREMENT
	dev_num	TINYINT UNSIGNED

cpu		
Key	id	AUTO_INCREMENT
	manufacturer	VARCHAR(16)
	cpu_model	VARCHAR(16)
	cpu_profile	TINYINT UNSIGNED
	cores	TINYINT UNSIGNED

cpu_type		
Key	id	AUTO_INCREMENT
	emulation	VARCHAR(16)

Figure 10. Relational Schema Diagram—View 2 of 4.

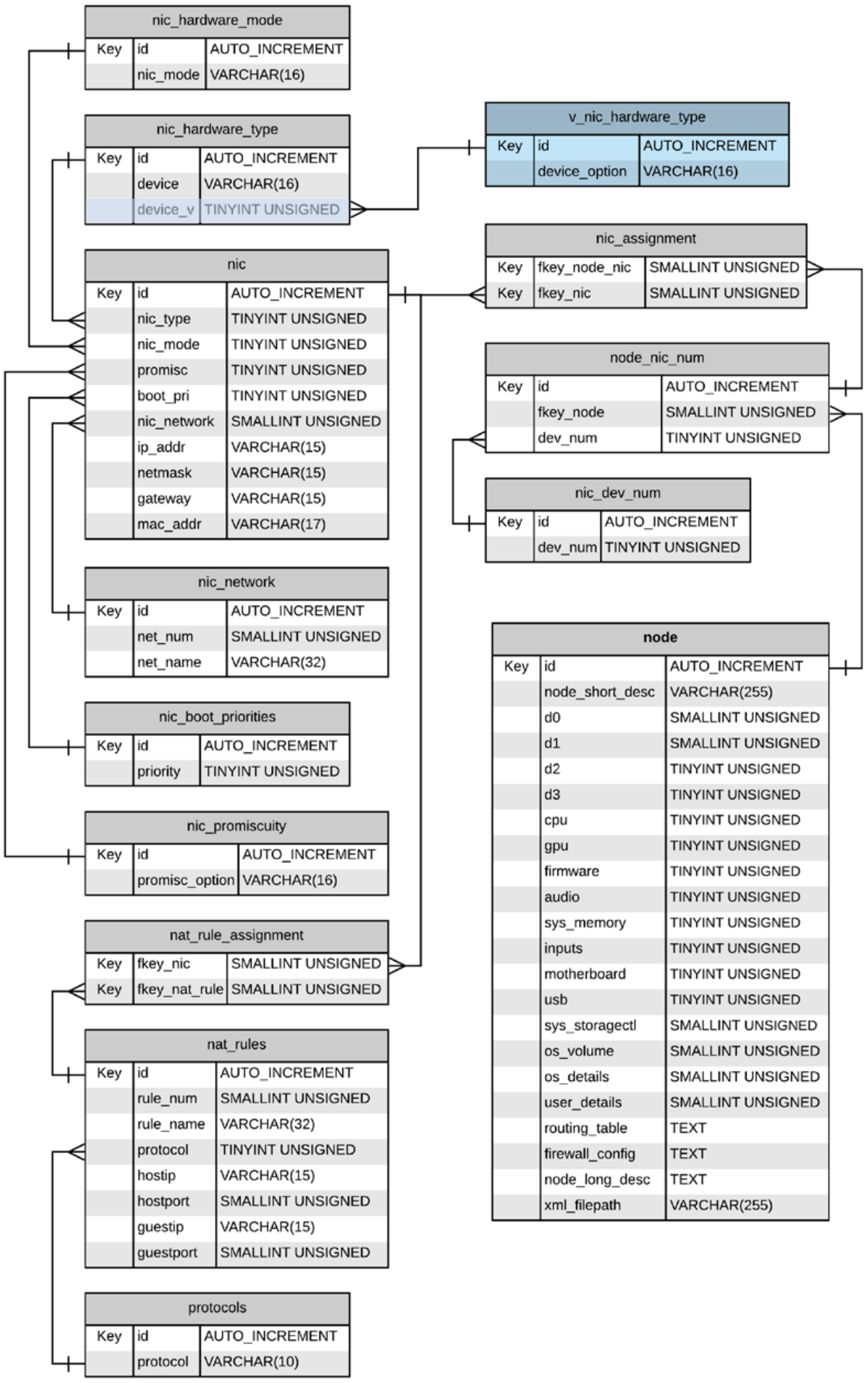


Figure 11. Relational Schema Diagram—View 3 of 4.

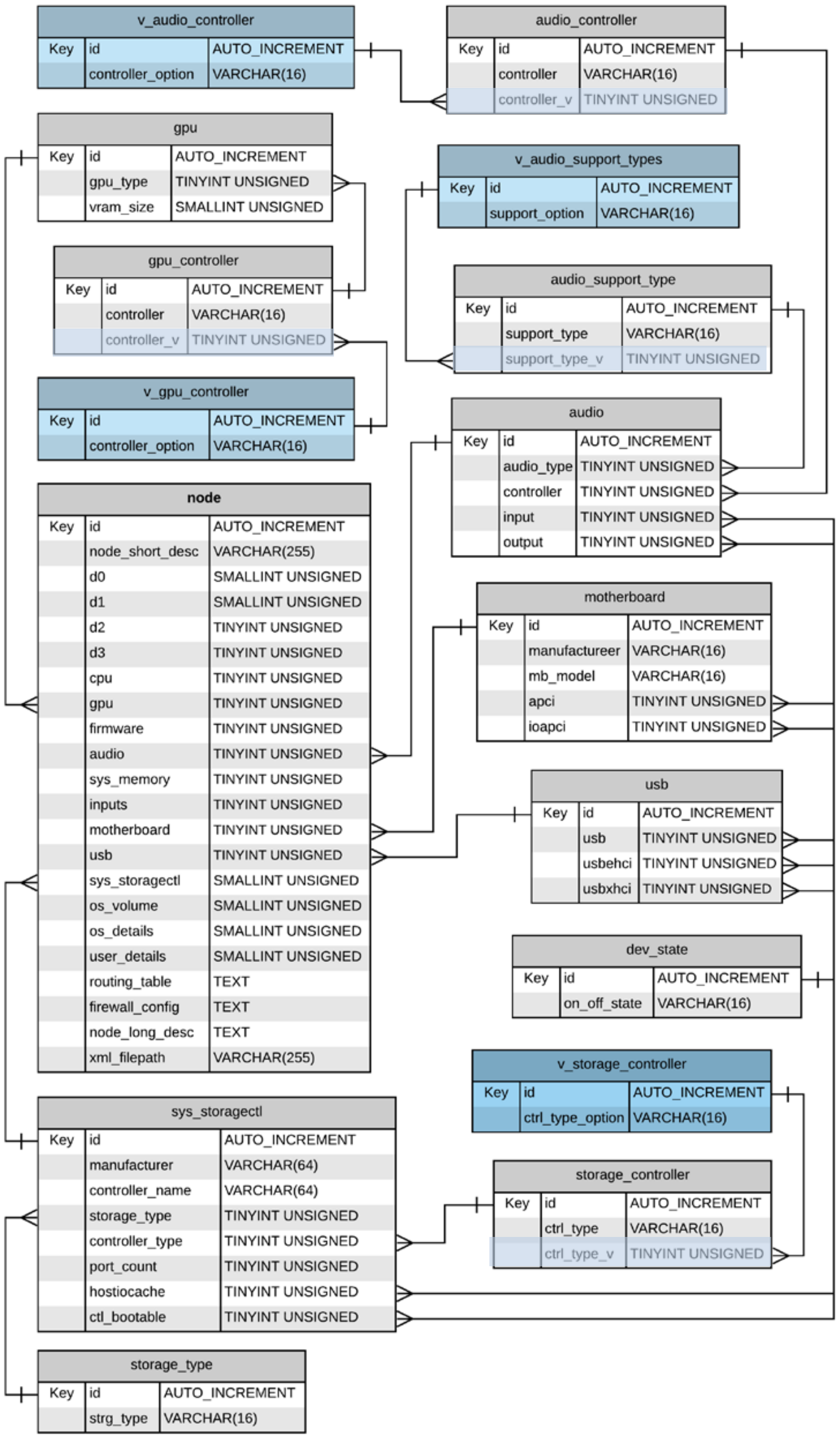

v_audio_controller
Key | id | AUTO_INCREMENT
controller_option | VARCHAR(16)
audio_controller
Key | id | AUTO_INCREMENT
controller | VARCHAR(16)
controller_v | TINYINT UNSIGNED
gpu
Key | id | AUTO_INCREMENT
gpu_type | TINYINT UNSIGNED
vram_size | SMALLINT UNSIGNED
v_audio_support_types
Key | id | AUTO_INCREMENT
support_option | VARCHAR(16)
gpu_controller
Key | id | AUTO_INCREMENT
controller | VARCHAR(16)
controller_v | TINYINT UNSIGNED
audio_support_type
Key | id | AUTO_INCREMENT
support_type | VARCHAR(16)
support_type_v | TINYINT UNSIGNED
v_gpu_controller
Key | id | AUTO_INCREMENT
controller_option | VARCHAR(16)
audio
Key | id | AUTO_INCREMENT
audio_type | TINYINT UNSIGNED
controller | TINYINT UNSIGNED
input | TINYINT UNSIGNED
output | TINYINT UNSIGNED
node
Key | id | AUTO_INCREMENT
node_short_desc | VARCHAR(255)
d0 | SMALLINT UNSIGNED
d1 | SMALLINT UNSIGNED
d2 | TINYINT UNSIGNED
d3 | TINYINT UNSIGNED
cpu | TINYINT UNSIGNED
gpu | TINYINT UNSIGNED
firmware | TINYINT UNSIGNED
audio | TINYINT UNSIGNED
sys_memory | TINYINT UNSIGNED
inputs | TINYINT UNSIGNED
motherboard | TINYINT UNSIGNED
usb | TINYINT UNSIGNED
sys_storagectl | SMALLINT UNSIGNED
os_volume | SMALLINT UNSIGNED
os_details | SMALLINT UNSIGNED
user_details | SMALLINT UNSIGNED
routing_table | TEXT
firewall_config | TEXT
node_long_desc | TEXT
xml_filepath | VARCHAR(255)
motherboard
Key | id | AUTO_INCREMENT
manufactureer | VARCHAR(16)
mb_model | VARCHAR(16)
apci | TINYINT UNSIGNED
ioapci | TINYINT UNSIGNED
usb
Key | id | AUTO_INCREMENT
usb | TINYINT UNSIGNED
usbehci | TINYINT UNSIGNED
usbxhci | TINYINT UNSIGNED
dev_state
Key | id | AUTO_INCREMENT
on_off_state | VARCHAR(16)
v_storage_controller
Key | id | AUTO_INCREMENT
ctrl_type_option | VARCHAR(16)
sys_storagectl
Key | id | AUTO_INCREMENT
manufacturer | VARCHAR(64)
controller_name | VARCHAR(64)
storage_type | TINYINT UNSIGNED
controller_type | TINYINT UNSIGNED
port_count | TINYINT UNSIGNED
hostiocache | TINYINT UNSIGNED
ctl_bootable | TINYINT UNSIGNED
storage_controller
Key | id | AUTO_INCREMENT
ctrl_type | VARCHAR(16)
ctrl_type_v | TINYINT UNSIGNED
storage_type
Key | id | AUTO_INCREMENT
strg_type | VARCHAR(16)

Figure 12. Relational Schema Diagram—View 4 of 4.

APPENDIX C. MAVNATT SCHEMA DESCRIPTION

Note: The schema description that follows refers to, but does not describe, the lookup tables used to supply options to the parent table. All lookup tables have an 'id' field for their first field and then the option criteria as the second field. Refer to the SQL code (Appendix D) for additional table and field details.

a. Table: Node

This table is central to the schema and provides access to all of the details of a node. It is expected that various nodes will have hardware and software configurations in common between them. To efficiently make use of database storage space and clarify the schema, only data that is unique to the node is stored within the table. All of the other fields within this table reference other tables so that many nodes may reuse the information. Unique constraints can be placed on the 'node' field alone to ensure the node does not become duplicated, but could also be tied to fields 'd0' and 'd1' in the event nodal names from the topological maps were simply copied over vice uniquely modified.

Fields

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED because MAVNATT is expected to handle more than 255 nodes.

2. **node_short_desc** (A#:47): text-based name of the node from 'node id' in the GraphML file (maximum of 255 characters).

3. **d0**: y coordinate from the GraphML file.

4. **d1**: x coordinate from the GraphML file.

5. **d2**: device type from the GraphML file; looks up the node's basic device type from the table: 'device_type'.

6. **d3**: operating system description from the GraphML file; stored in referenced table 'os_description'.

7. **cpu**: nodes’s VirtualBox (VBox) cpu profile; stored in referenced table ‘cpu’.

8. **gpu**: nodes’s VBox graphics profile; stored in referenced table ‘gpu’.

9. **firmware** (A#:9): node’s VBox firmware profile; stored in referenced table ‘firmware’.

10. **audio**: node’s VBox audio profile; stored in referenced table ‘audio’.

11. **sys_memory**: node’s VBox memory profile; stored in referenced table ‘sys_memory’.

12. **inputs**: node’s VBox input profile; stored in referenced table ‘inputs’.

13. **motherboard**: node’s VBox motherboard profile; stored in referenced table ‘motherboard’.

14. **usb**: node’s VBox device usb profile; stored in referenced table ‘usb’.

15. **sys_storageclt**: node’s VBox drive storage controller profile; stored in referenced table ‘sys_storageclt’.

16. **os_volume**: OS storage volume details specific to the node; stored in referenced table ‘os_volume’.

17. **os_details**: OS details specific to the node; stored in referenced table ‘os_details’.

18. **user_details**: user account details specific to the node; stored in referenced table ‘user_details’.

19. **routing_table** (A#:31): node’s networking routing table file contents (maximum of 65,535 characters).

20. **firewall_config** (A#:32): node’s networking firewall configuration file contents (maximum of 65,535 characters).

21. **node_long_desc** (A#:48): long detailed description of the node (maximum of 65,535 characters).

22. **xml_filepath**: location of the XML file containing all of the settings for the VM (maximum of 255 characters).

b. Table: Topology

The topology table contains the individual portions of the network topology for each node; derived from the GraphML file.

Fields

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED because MAVNATT is not expected to scan very large networks greater than 10,000 nodes.

2. **edge_source**: refers to a node and looks this up from the table: 'node'.

3. **target_node**: refers to a node and looks this up from the table: 'node'.

4. **d6**: the IP address of the node, limited to 15 characters.

5. **d7**: the destination interface for the node, limited to 64 characters.

6. **d8**: the IP address of the destination, limited to 15 characters.

7. **d9**: the nodal interface used to connect to the distant interface (**d7)**.

c. Table: os_description

The OS Description table contains two text fields and is used to store the basic description of the OS and the location of the image that can build a node. Recommend using 'd10' to constrain the uniqueness of the entry.

Fields

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the expected amount of operating systems within the network is expected to be less than 255.

2. **d3**: textual description of the node's operating system; from the GraphML file. Expected string length is 64 characters.

3. **d10**: location of the image of the operating system which can build the node; from the GraphML file. Expected string length is less than 255 characters.

4. **version** (A#:46): the specific version number of the OS; field length is 64 characters.

d. Table: cpu

The cpu table references one supporting table to ensure that additional options which VBox cannot recognize are not accidentally added to the database.

Fields

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the expected variance of CPUs is expected to be less than 255 on any given tactical network.

2. **manufacturer** (A#:7): the manufacturer of the CPU.

3. **cpu_model** (A#:8): the model of the CPU.

4. **cpu_profile** (A#:6): the CPU emulation mode for the node; looks up available entries from table: 'cpu_type'.

5. **cores** (A#:5): the number of virtual CPUs available to the node. For VirtualBox v6, the maximum is 32 CPUs.

e. Table Group: gpu

The GPU table references one supporting table to ensure that additional options which VBox cannot recognize are not accidentally added to the database.

Fields for Table: gpu

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the expected amount of graphics configurations is expected to be less than 255.

2. **gpu_type** (A#:12): the graphics controller type VBox should emulate for the node; looks up available entries from table: ‘gpu_controller’. 4 current options: none|vboxvga|vmsvga|vboxsvga.

3. **vram_size** (A#:11): the amount of video RAM in MB to allocated to the node.

Fields for Table: gpu_controller

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the expected amount of graphics configurations is expected to be less than 255.

2. **controller** (A#:12): the audio hardware controller type, 16 characters. This field is empty in physical database, pre-filled in the virtual database, and meant to hold the corresponding physical database value in the baseline database.

3. **controller_v (baseline ONLY)** (A#:12): the audio hardware controller types recognized by VBox. This field looks up from table: ‘v_gpu_controller’.

Note: Fields 2 and 3 of this table in the baseline database combine to form a unique field to prevent duplicate entries.

f. Table: firmware

This table is a lookup for the table: ‘node’ to ensure that only the necessary options which VBox can recognize are available.

Fields

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since there are only four options.

2. **firm_type** (A#:9): specifies the firmware type VBox should emulate for the node.

g. Table Group: Audio

The Audio table references three supporting tables to reduce data redundancy and ensure that additional options which VBox cannot recognize are not accidentally added to the database.

Fields for Table: audio

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the expected amount of audio type available to VirtualBox is currently seven.

2. **audio_type** (A#:1): the audio support to be given to the node from VBox; looks up available entries from table: ‘audio_support_type’.

3. **controller** (A#:2): the audio controller used for the audio; looks up available entries from table: ‘audio_controller’.

4. **input** (A#:3): turns audio capture on or off for the VM; looks up available entries from table: ‘dev_state’.

5. **output** (A#:4): turns audio playback on or off for the VM; looks up available entries from table: ‘dev_state’.

Fields for Table: audio_support_type

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the amount of audio support types is seven.

2. **support_type** (A#:1): the audio hardware support type, 16 characters. This field is empty in the physical database, pre-filled in the virtual database, and meant to hold the corresponding physical database value in the baseline database.

3. **support_type_v (baseline ONLY)** (A#:1): one of the seven audio hardware support types recognized by VBox. This field looks up from table: ‘v_audio_support_types’.

Note: Fields 2 and 3 of this table in the baseline database combine to form a unique field to prevent duplicate entries.

Fields for Table: audio_controller

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the amount of audio support types is three.

2. **controller** (A#:2): the audio hardware controller type, 16 characters. This field is empty in the physical database, pre-filled in the virtual database, and meant to hold the corresponding physical database value in the baseline database.

3. **controller_v (baseline ONLY)** (A#:2): one of the three audio hardware controller options recognized by VBox. This field looks up from table: ‘v_audio_controller’.

Note: Fields 2 and 3 of this table in the baseline database combine to form a unique field to prevent duplicate entries.

h. Table: sys_memory

The System Memory table contains several attributes of the node's memory.

Fields

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the expected amount of memory configurations on the network is less than 255.

2. **manufacturer** (A#:15): the name of the memory vendor.

3. **mem_model** (A#:16): the model of the memory.

4. **mem_size** (A#:17): the amount of memory in MB that is allocated to the node.

i. Table Group: Inputs

The Inputs table group references two supporting tables to ensure that only the necessary options which VBox can recognize are available.

Fields

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the number of input combinations is less than 255.

2. **keyboard** (A#:14): the keyboard device VBox should emulate for the node; looks up available entries from table: 'keyboard_dev'.

3. **mouse** (A#:13): the mouse device VBox should emulate for the node; looks up available entries from table: 'mouse_dev'.

j. *Table: motherboard*

The Motherboard table references one other table to ensure that only the necessary options which VBox can recognize are available.

Fields

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the number of motherboard combinations is less than 255.

2. **manufacturer** (A#:18): name of the motherboard manufacturer.

3. **mb_model** (A#:19): motherboard model.

4. **apci** (A#:20): enables APCI support for the node and is required for 64-bit operating systems; looks up available entries from table: ‘dev_state’.

5. **ioapci** (A#:21): enables I/O APCI support for the node and is required for 64-bit operating systems; looks up available entries from table: ‘dev_state’.

k. *Table Group: usb*

The USB table specifies which USB controllers are available to the node.

Fields

1. **id**: auto incremented, primary key. Data type TINYINT UNSIGNED since the number of option combinations is less than 255.

2. **usb** (A#:63): turns on or off the USB controller to the node; looks up available states from table: ‘dev_state’.

3. **usbehci** (A#:64): turns on or off the USBEHCI controller to the node; looks up available states from table: ‘dev_state’.

4. **usbxhci** (A#:65): turns on or off the USBXHCI controller to the node; looks up available states from table: ‘dev_state’.

l. Table Group: System Storage Controller

The System Storage Controller table group contains the details for the configurations of the node’s storage controllers for all storage devices.

Fields

1. **id**: auto incremented, primary key. Data type SMALLINT UNSIGNED since the unique configurations per node may be large.

2. **manufacturer** (A#:56): the manufacturer of the storage controller.

3. **controller_name** (A#:57): the name of the storage controller; indexed to ensure required uniqueness for VBox.

4. **storage_type** (A#:58): the type of storage device VBox should emulated for the VM; looks up available entries from table: ‘storage_type’.

5. **controller_type** (A#:59): the type of storage controller VBox should emulate for the VM; looks up available entries from table: ‘storage_controller’.

6. **port_count** (A#:60): the amount of ports that the controller should have.

7. **hostiocache** (A#:61): host IO cache on or off.

8. **bootable_ctl** (A#:62): allows the controller to be used as a boot option.

Fields for Table: controller_type

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the number of storage controller types for VBox is ten.

2. **device** (A#:59): the storage controller type, 16 characters. This is empty in the physical database, pre-filled in the virtual database, and meant to hold the corresponding physical database value in the baseline database.
3. **device_v** (A#:59): one of the ten storage controller options recognized by VBox. This field looks up from table: ‘v_storage_controller’.

Note: Fields 2 and 3 of this table in the baseline database combine to form a unique field to prevent duplicate entries.

m. Table: os_volume

The OS Volume table contains the details about the storage volumes specific to the node’s OS.

1. **id**: auto incremented, primary key. Data type SMALLINT UNSIGNED since the number of expected volumes may be large, depending on the accuracy of the virtualization.
2. **volume_path** (A#:49): logical volume path (64 characters)
3. **volume_name** (A#:50): human readable volume name (64 characters)
4. **volume_size** (A#:51): the size, in MB, the OS should allocate to the virtual storage device assigned to the VM; uses MEDIUMINT UNSIGNED data type.

n. Table Group: OS Details

The OS Details table group contains the details specific to a particular node’s operating system. Row entries may all be unique.

Fields

1. **id**: auto incremented, primary key. Data type SMALLINT UNSIGNED since the number of expected configurations should be no greater than the amount of expected nodes within the network.

2. **product_key** (A#:38): the product key.

3. **locale** (A#:39): the base locale for the node (ex: en_US).

4. **country** (A#:40): the two letter country code for the node.

5. **time_zone** (A#:41): time_zone for the node.

6. **hostname** (A#:42): hostname for the node.

7. **swap_size** (A#:43): swap partition size for the node's operating system. Data type is MEDIUMINT UNSIGNED to provide ample allocation.

8. **frontend** (A#:44): used to specify the frontend that the node's operating system will use.

9. **startup_apps** (A#:45): list of the startup apps for the node; looks up from table: 'startup_apps' as the list may be identical across several nodes. Data type is SMALLINT UNSIGNED in case every node has a unique set of startup apps.

o. Table: user_details

The User Details table contains the details specific to a particular node's user account.

Fields

1. **id**: auto incremented, primary key. Data type SMALLINT UNSIGNED since the number of expected configurations should be no greater than the number of expected nodes within the network.

2. **login** (A#:35): the login of the user for the node; set to a length of 24 characters.

3. **password** (A#:36): the password for the user's login for the node; set to a length of 32 characters.

4. **user_name** (A#:37): the full name on the node's user account; set to a length of 48 characters.

p. Table Group: Boot Device Order

The Boot Device Order table group is meant to provide a way to set the order in which the node tries at most four unique devices upon boot up and has no 'id' field. This process starts with ensuring unique combinations of the 'fkey_node' and 'dev_pri' fields within table: 'boot_priorities'. 'fkey_node' references to table 'node', ensuring that a boot priority is assigned to a preexisting node. To ensure that only four priorities (1-4) are selectable, the 'dev_pri' field references a table that only has those values available. Table: 'boot_order' can then ensure unique combinations of 'id' from 'boot_priorities' and 'device' from table: 'boot_device' to complete the boot order.

Fields for Table: 'boot_priorities'

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number of node-boot device combinations is expected to be less than 65,535 for 10,000 nodes.

2. **fkey_node**: the 'id' of the node; referenced from table: 'node'.

3. **dev_pri** (A#:10): contains a priority number; looks up from table: 'priorities'

Unique combinations of the fields (fkey_node, dev_pri) are used to ensure only one priority is assigned to a node.

Fields for Table: 'boot_order'

1. **fkey_boot_priority**: refers to ‘id’ field in table: ‘bootPriorities’.

2. **fkey_boot_device**: contains the device to assign to a boot priority; looks up from table: ‘boot_device’ (A#:10).

These two fields together are used as a primary key to ensure only unique combinations are allowed.

q. Table Group: NIC Assignments

The Network Interface Card (NIC) Assignment table group associates a node with up to eight NICs, which is the maximum number that VirtualBox will support per VM. This process of associating eight unique NICs with a particular node starts with ensuring unique combinations of the ‘fkey_node’ and ‘dev_num’ fields within table: ‘node_nic_num’. ‘fkey_node’ references to table ‘node’, ensuring that the NIC is assigned to a preexisting node. To ensure that only eight NICs (numbers 1–8) can be assigned, the ‘dev_num’ field references a table that only has those values available. Table: ‘nic_assignment’ can then ensure unique combinations of the ‘id’ fields from table ‘node_nic_num’ and table ‘nic’ to complete the assignment.

Each NIC can also have network address translation (NAT) rules assigned to it. A NIC is tied to multiple NAT rules via junction table ‘nat_rule_assignment’, which contains unique combinations of the foreign keys that reference the ‘id’ fields of tables ‘nic’ and ‘nat_rules’.

Fields for Junction Table: ‘nic_assignment’

1. **fkey_node_nic**: Contains a foreign key referencing the ‘id’ of table ‘node_nic_num’.

2. **fkey_nic**: Contains a foreign key referencing the ‘id’ of table ‘nic’.

These two fields together are used as a primary key to ensure only unique combinations are allowed.

Fields for junction table: ‘node_nic_num’

1. **fkey_node**: Contains a foreign key referencing the ‘id’ of the node from table: ‘node’.

2. **dev_num** (A#:22): the single digit assignment (1-8) of the NIC to the node; looks up from ‘dev_num’ from table: ‘nic_dev_num’.

These two fields together are used as a constraint to ensure that a node can only have eight unique NICs.

Fields for Table: ‘nic’

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number of uniquely defined NICs within the network is expected to be more than 255 for 1,000 nodes.

2. **nic_type** (A#:23): the NIC type for VBox to emulate; looks up from ‘device’ of table: ‘nic_hardware_type’.

3. **nic_mode** (A#:24): the networking mode assigned to the NIC; looks up from ‘mode’ of table: ‘nic_hardware_mode’.

4. **promisc** (A#:25): promiscuity option for the NIC; looks up from ‘option’ of table: ‘nic_promiscuity’.

5. **boot_pri** (A#:26): a single digit number indicating network boot priority; looks up from ‘priority’ of table: ‘nic_boot_priorities’.

6. **nic_network**: the network the NIC is to connect to; details stored in table ‘nic_network’.

7. **ip_addr** (A#:27): the IP Address assigned to the NIC.

8. **netmask** (A#:28): the subnet mask of the IP Address.

9. **gateway** (A#:29): the gateway the NIC should use.

10. **mac_addr** (A#:30): the MAC address assigned to the NIC.

Fields for Table: 'nic_hardware_type'

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the number of NIC hardware types for VBox is six.
2. **device** (A#:23): the NIC hardware controller, 16 characters. This is empty in the physical database, pre-filled in the virtual database, and meant to hold the corresponding physical database value in the baseline database.
3. **device_v** (A#:23): one of the six NIC hardware controller options recognized by VBox. This field looks up from table: 'v_nic_hardware_type'.

Note: Fields 2 and 3 of this table in the baseline database combine to form a unique field to prevent duplicate entries.

Fields for Table: 'nic_network'

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number network assignments is expected to be less than 65,535 for 1,000 nodes.
2. **net_num** (A#:33): the number VBox uses to refer to a network name.
3. **net_name** (A#:33): the network name the NIC is assigned to, which can be either a VBox internal network or a NAT network. The name itself can be expected to be an IP address or a user defined name.

Note: This table could be optimized to record unique sets of 'net_num' and 'net_name' if 'net_name' was constrained to be unique and then 'net_num' and 'net_name' pairs were constrained to be unique, but is overly complex for this application. It may become necessary if MAVNATT is expected to handle a larger or more complex network.

Fields for Junction Table: 'nat_rule_assignment'

1. **fkey_nic**: Contains a foreign key referencing the 'id' of table 'nic'.

2. **fkey_nat_rule**: Contains a foreign key referencing the ‘id’ of table ‘nat_rules’.

These two fields together are used as a primary key to ensure only unique combinations are allowed.

Fields for Table: ‘nat_rules’ (A#:34)

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number network assignments is expected to be less than 65,535 for 1,000 nodes.

2. **rule_num**: the number VBox uses to refer to a NAT rule. Data type is SMALLINT UNSIGNED since the number NAT rules is expected to be less 65,535.

3. **rule_name**: the user assigned name to the rule; 32 characters.

4. **protocol**: TCP or UDP protocol; looks up from ‘protocol’ field of table ‘protocols’.

5. **hostip**: the optional host IP address the rule will apply to.

6. **hostport**: the host port traffic will be coming from.

7. **guestip**: the guest IP address where traffic will be sent to.

8. **guestport**: the guest port traffic will be sent to.

r. Table Group: Serial Ports

The Serial Ports table group associates the node with at most four serial ports as that is all VirtualBox supports per VM. This process starts with ensuring unique combinations of the ‘fkey_node’ and ‘dev_num’ fields within table: ‘node_serial_ports’. ‘fkey_node’ references to table ‘node’, ensuring that a serial port is assigned to a preexisting node. To ensure that only four serial ports can be assigned, the ‘dev_num’ field references table: ‘serial_dev_num’ which only has values “1,2,3,4” available. Table:

‘serial_ports’ can then ensure unique combinations of ‘id’ from table: ‘node_serial_ports’ and ‘id’ from table: ‘serial_port_config’ to complete the serial port assignment.

Fields for Table: ‘node_serial_ports’

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number serial port assignments is expected to be less than 65,535 for 1,000 nodes.

2. **fkey_node**: Contains a foreign key referencing the ‘id’ of the node from table: ‘node’.

3. **dev_num**: contains a number; looks up from table: ‘serial_dev_num’.

Fields for Table: ‘serial_ports’

1. **id:** auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number serial port assignments is expected to be less than 65,535 for 1,000 nodes.

2. **fkey_node_serial**: refers to ‘id’ field in table: ‘node_serial_ports’.

3. **fkey_port_config**: contains the device configuration to assign to a node’s serial port; looks up from table: ‘serial_port_config’.

Fields for Table: ‘serial_port_config’

1. **id**: auto incremented, primary key. Data type is SMALLINT UNSIGNED since the number serial port assignments is expected to be less than 65,535 for 1,000 nodes.

2. **io_base** (A#:52): I/O base number for the port configuration.

3. **irq** (A#:52): interrupt number for the port configuration.

4. **serial_mode** (A#:53): argument for the VBox modifyvm --uartmode attribute.

s. Table Group: LPT Ports

The LPT Ports table group associates the node with all of the parallel ports that pertain to it. VirtualBox provides access to the parallel ports on the host to the VM, which means that the VM can have as many parallel ports as are available.

Fields for Table 'lpt'

1. **fkey_node**: Contains a foreign key referencing the 'id' of the node from table: 'node'.

2. **fkey_lpt_config**: Contains a foreign key referencing the 'id' from the table: 'lpt_config'.

Fields for Table 'lpt_config'

1. **id**: auto incremented, primary key. Data type is TINYINT UNSIGNED since the number of expected LPT configurations is less than 255.

2. **lpt_mode** (A#:54): the host's device name of the parallel port to use. 64 characters have been initially allocated for the value.

3. **io_base** (A#:55): a hexadecimal value for the I/O Base used to access the parallel port.

4. **irq** (A#:55): a single digit decimal number for the interrupt used to access the parallel port.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. MAVNATT SCHEMA SQL CODE

A. SCHEMA BUILD CODE

This code is intended to be used to create the physical, baseline, and virtual databases with in MAVNATT's relational database system. The following commands must be used prior to running the schema generation code below for each database:

```
CREATE DATABASE physical;
USE physical;
< SQL Schema Generation Code (Section B) >

CREATE DATABASE baseline;
USE baseline;
< SQL Schema Generation Code (Section B) >
< Baseline Schema Alteration Code (Section C) >
< Baseline Database Lookup Table Insertion Code (Section D) >

CREATE DATABASE virtual;
USE virtual;
< SQL Schema Generation Code (Section B) >
< Virtual Database Lookup Table Insertion Code (Section E) >
```

Note: Two dashes are used to make comments in SQL.

B. SQL SCHEMA GENERATION CODE

```
-- 1 Device Type Table
CREATE TABLE device_type (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    d2 VARCHAR(16), -- node device types
    PRIMARY KEY (id)
);

-- 1 OS Description Table
CREATE TABLE os_description ( -- supplies node(d3)
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    d3 VARCHAR(64),
    d10 VARCHAR(255),
    os_version VARCHAR(64),
    PRIMARY KEY (id)
);

-- 1 Firmware Table
CREATE TABLE firmware (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    firm_type VARCHAR(16),
    PRIMARY KEY (id)
);
```

```
-- 1 System Memory Table
CREATE TABLE sys_memory (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    manufacturer VARCHAR(16),
    mem_model VARCHAR(16),
    mem_size MEDIUMINT UNSIGNED,
    PRIMARY KEY (id)
);

-- 3 Input Tables: keyboard_dev, mouse_dev, inputs
CREATE TABLE keyboard_dev (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    keyb_type VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE mouse_dev (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    mouse_type VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE inputs (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    keyboard TINYINT UNSIGNED,
    mouse TINYINT UNSIGNED,
    FOREIGN KEY (keyboard) REFERENCES keyboard_dev(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (mouse) REFERENCES mouse_dev(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- 2 CPU Tables: cpu_type, cpu
CREATE TABLE cpu_type (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    emulation VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE cpu (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    manufacturer VARCHAR(16),
    cpu_model VARCHAR(16),
    cpu_profile TINYINT UNSIGNED,
    cores TINYINT UNSIGNED,   -- limited to 32 cores in
VirtualBox
    FOREIGN KEY (cpu_profile) REFERENCES cpu_type(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- 2 GPU Tables: gpu_controller, gpu
CREATE TABLE gpu_controller (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    controller VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE gpu (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    gpu_type TINYINT UNSIGNED,
    vram_size TINYINT UNSIGNED,
    FOREIGN KEY (gpu_type) REFERENCES gpu_controller(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- State Table: REFERENCED by Audio, Motherboard, and USB
branches
-- MUST be created prior to the others!
CREATE TABLE dev_state (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    on_off_state VARCHAR(16),
    PRIMARY KEY (id)
);

-- 3 Audio Tables: audio_support_type, audio_controller, audio
CREATE TABLE audio_support_type (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    support_type VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE audio_controller (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    controller VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE audio (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    audio_type TINYINT UNSIGNED,
    controller TINYINT UNSIGNED,
    input TINYINT UNSIGNED,
    output TINYINT UNSIGNED,
    FOREIGN KEY (audio_type) REFERENCES audio_support_type(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (controller) REFERENCES audio_controller(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (input) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (output) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- 1 Motherboard Table
CREATE TABLE motherboard (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    manufacturer VARCHAR(16),
    mb_model VARCHAR(16),
    apci TINYINT UNSIGNED,
    ioapci TINYINT UNSIGNED,
    FOREIGN KEY (apci) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (ioapci) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- 1 USB Table
CREATE TABLE usb (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    usb TINYINT UNSIGNED,
    usbehci TINYINT UNSIGNED,
    usbxhci TINYINT UNSIGNED,
    FOREIGN KEY (usb) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (usbehci) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (usbxhci) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- 3 Storage Controller Tables: storage_type, storage_controller,
-- sys_storagectl
CREATE TABLE storage_type (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    strg_type VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE storage_controller (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    ctrl_type VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE sys_storagectl (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    manufacturer VARCHAR(64),
    controller_name VARCHAR(64) NOT NULL,
    storage_type TINYINT UNSIGNED,
    controller_type TINYINT UNSIGNED,
    port_count TINYINT UNSIGNED, -- limited to range of 1–30
    hostiocache TINYINT UNSIGNED,
    ctl_bootable TINYINT UNSIGNED,
    FOREIGN KEY (storage_type) REFERENCES storage_type(id) ON
UPDATE CASCADE,
    FOREIGN KEY (controller_type) REFERENCES
storage_controller(id) ON UPDATE CASCADE,
    FOREIGN KEY (hostiocache) REFERENCES dev_state(id)
```

```
      ON UPDATE CASCADE,
    FOREIGN KEY (ctl_bootable) REFERENCES dev_state(id)
      ON UPDATE CASCADE,
    CONSTRAINT storagectl_controller_name UNIQUE
(controller_name),
    PRIMARY KEY (id)
);

-- 1 OS Volume Details Table
CREATE TABLE os_volume (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    volume_path VARCHAR(64),
    volume_name VARCHAR(64),
    volume_size MEDIUMINT UNSIGNED,
    PRIMARY KEY (id)
);

-- 1 User Details Table
CREATE TABLE user_details (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    login VARCHAR(24),
    password VARCHAR(32),
    user_name VARCHAR(48),
    PRIMARY KEY (id)
);

-- 2 OS Detail Tables: startup_apps, os_details
CREATE TABLE startup_apps (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    app_list VARCHAR(255),
    PRIMARY KEY (id)
);

CREATE TABLE os_details (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    product_key VARCHAR(255),
    locale VARCHAR(10),
    country VARCHAR(2),
    time_zone VARCHAR(16),
    hostname VARCHAR(255),
    swap_size MEDIUMINT UNSIGNED,
    frontend VARCHAR(64),
    startup_apps SMALLINT UNSIGNED,
    FOREIGN KEY (startup_apps) REFERENCES startup_apps(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (id)
);

-- 1 Node Table
CREATE TABLE node (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    node_short_desc VARCHAR(255),
    d0 SMALLINT UNSIGNED, -- y coord
    d1 SMALLINT UNSIGNED, -- x coord
    d2 TINYINT UNSIGNED,
    d3 TINYINT UNSIGNED,
    cpu TINYINT UNSIGNED,
    gpu TINYINT UNSIGNED,
    firmware TINYINT UNSIGNED,
    audio TINYINT UNSIGNED,
    sys_memory TINYINT UNSIGNED,
    inputs TINYINT UNSIGNED,
    motherboard TINYINT UNSIGNED,
    usb TINYINT UNSIGNED,
    sys_storagectl SMALLINT UNSIGNED,
    os_volume SMALLINT UNSIGNED,
    os_details SMALLINT UNSIGNED,
    user_details SMALLINT UNSIGNED,
    routing_table TEXT,
    firewall_config TEXT,
    node_long_desc TEXT,
    xml_filepath VARCHAR(255),
    FOREIGN KEY (d2) REFERENCES device_type(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (d3) REFERENCES os_description(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (cpu) REFERENCES cpu(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (gpu) REFERENCES gpu(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (firmware) REFERENCES firmware(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (audio) REFERENCES audio(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (sys_memory) REFERENCES sys_memory(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (inputs) REFERENCES inputs(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (motherboard) REFERENCES motherboard(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (usb) REFERENCES usb(id) ON UPDATE CASCADE
      ON DELETE CASCADE,
    FOREIGN KEY (sys_storagectl) REFERENCES sys_storagectl(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (os_volume) REFERENCES os_volume(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (os_details) REFERENCES os_details(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (user_details) REFERENCES user_details(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (id)
);

-- 1 Topology Table
CREATE TABLE topology (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    edge_source SMALLINT UNSIGNED NOT NULL,
    target_node SMALLINT UNSIGNED NOT NULL,
    d6 VARCHAR(15), -- node IP address
    d7 VARCHAR(64), -- destination interface
    d8 VARCHAR(15), -- destination IP address
    d9 VARCHAR(64), -- node’s interface
    FOREIGN KEY (edge_source) REFERENCES node(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (target_node) REFERENCES node(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

-- 4 Boot Order Tables: priorities, boot_device, boot_priorities,
-- boot_order
CREATE TABLE priorities (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    priority TINYINT UNSIGNED,
    PRIMARY KEY (id)
);

CREATE TABLE boot_device (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    device VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE boot_priorities (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    fkey_node SMALLINT UNSIGNED NOT NULL,
    dev_pri TINYINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_node) REFERENCES node(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (dev_pri) REFERENCES priorities(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

CREATE TABLE boot_order (
    fkey_boot_priority SMALLINT UNSIGNED NOT NULL,
    fkey_boot_device TINYINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_boot_priority) REFERENCES
boot_priorities(id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (fkey_boot_device) REFERENCES boot_device(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (fkey_boot_priority, fkey_boot_device)
);

-- 4 Serial Port Tables: serial_dev_num, serial_port_config,
node_serial_ports, serial_ports
CREATE TABLE serial_dev_num (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    dev_num TINYINT UNSIGNED,
    PRIMARY KEY (id)
);

CREATE TABLE serial_port_config (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    io_base VARCHAR(5),
    irq TINYINT UNSIGNED,
    serial_mode VARCHAR(128),
    PRIMARY KEY (id)
);

CREATE TABLE node_serial_ports (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    fkey_node SMALLINT UNSIGNED NOT NULL,
    dev_num TINYINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_node) REFERENCES node(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (dev_num) REFERENCES serial_dev_num(id)
      ON UPDATE CASCADE,
    CONSTRAINT uc_node_serial UNIQUE (fkey_node, dev_num),
    PRIMARY KEY (id)
);

CREATE TABLE serial_ports (
    fkey_node_serial SMALLINT UNSIGNED NOT NULL,
    fkey_port_config SMALLINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_node_serial) REFERENCES
node_serial_ports(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (fkey_port_config) REFERENCES
serial_port_config(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (fkey_node_serial, fkey_port_config)
);

-- 2 LPT Port Tables: lpt_config, lpt
CREATE TABLE lpt_config (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    lpt_mode VARCHAR(64),
    io_base VARCHAR(5),
    irq TINYINT UNSIGNED,
    PRIMARY KEY (id)
);

CREATE TABLE lpt (
    fkey_node SMALLINT UNSIGNED NOT NULL,
    fkey_lpt_config TINYINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_node) REFERENCES node(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (fkey_lpt_config) REFERENCES lpt_config(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (fkey_node, fkey_lpt_config)
);

-- 8 NIC Tables: nic_dev_num, nic_hardware_type,
-- nic_hardware_mode, nic_promiscuity, nic_boot_priorities,
-- nic_network, nic, node_nic_num, nic_assignment

CREATE TABLE nic_dev_num (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    dev_num TINYINT UNSIGNED,
    PRIMARY KEY (id)
);

CREATE TABLE nic_hardware_type (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    device VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE nic_hardware_mode (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    nic_mode VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE nic_promiscuity (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    promisc_option VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE nic_boot_priorities (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    priority VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE nic_network (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    net_num TINYINT,
    net_name VARCHAR(32),
    PRIMARY KEY (id)
);

CREATE TABLE nic (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    nic_type TINYINT UNSIGNED,
    nic_mode TINYINT UNSIGNED,
    promisc TINYINT UNSIGNED,
    boot_pri TINYINT UNSIGNED,
    nic_network SMALLINT UNSIGNED,
    ip_addr VARCHAR(15),
    netmask VARCHAR(15),
    gateway VARCHAR(15),
    mac_addr VARCHAR(17),
    FOREIGN KEY (nic_type) REFERENCES nic_hardware_type(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (nic_mode) REFERENCES nic_hardware_mode(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (promisc) REFERENCES nic_promiscuity(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (boot_pri) REFERENCES nic_boot_priorities(id)
      ON UPDATE CASCADE,
    FOREIGN KEY (nic_network) REFERENCES nic_network(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (id)
);

CREATE TABLE node_nic_num (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    fkey_node SMALLINT UNSIGNED NOT NULL,
    dev_num TINYINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_node) REFERENCES node(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (dev_num) REFERENCES nic_dev_num(id)
      ON UPDATE CASCADE,
    CONSTRAINT uc_node_nic UNIQUE (fkey_node, dev_num),
    PRIMARY KEY (id)
);

CREATE TABLE nic_assignment (
    fkey_node_nic SMALLINT UNSIGNED NOT NULL,
    fkey_nic SMALLINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_node_nic) REFERENCES node_nic_num(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (fkey_nic) REFERENCES nic(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (fkey_node_nic, fkey_nic)
);

-- 3 NAT Tables: protocols, nat_rules, nat_rule_assignment
CREATE TABLE protocols (
    id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    protocol VARCHAR(10),
    PRIMARY KEY (id)
);

CREATE TABLE nat_rules (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    rule_num SMALLINT UNSIGNED,
    rule_name VARCHAR(32),
    protocol TINYINT UNSIGNED,
    hostip VARCHAR(15),
    hostport SMALLINT UNSIGNED,
    guestip VARCHAR(15),
    guestport SMALLINT UNSIGNED,
    FOREIGN KEY (protocol) REFERENCES protocols(id)
      ON UPDATE CASCADE,
    PRIMARY KEY (id)
);

CREATE TABLE nat_rule_assignment (
    fkey_nic SMALLINT UNSIGNED NOT NULL,
    fkey_nat_rule SMALLINT UNSIGNED NOT NULL,
    FOREIGN KEY (fkey_nic) REFERENCES nic(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (fkey_nat_rule) REFERENCES nat_rules(id)
      ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (fkey_nic, fkey_nat_rule)
);
```

C. BASELINE SCHEMA ALTERATION CODE

```
USE baseline;

CREATE TABLE v_gpu_controller (
     id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
     controller_option VARCHAR(16),
     PRIMARY KEY (id)
);

ALTER TABLE gpu_controller
     ADD COLUMN controller_v TINYINT UNSIGNED,
     ADD FOREIGN KEY (controller_v) REFERENCES
v_gpu_controller(id) ON UPDATE CASCADE,
     ADD CONSTRAINT uc_gpu_controller_phy_vir UNIQUE (controller,
controller_v);
--);

CREATE TABLE v_audio_support_types (
     id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
     support_option VARCHAR(16),
     PRIMARY KEY (id)
);

ALTER TABLE audio_support_type
     ADD COLUMN support_type_v TINYINT UNSIGNED,
     ADD FOREIGN KEY (support_type_v) REFERENCES
v_audio_support_types(id) ON UPDATE CASCADE,
     ADD CONSTRAINT uc_audio_support_phy_vir UNIQUE
(support_type, support_type_v);

CREATE TABLE v_audio_controller (
     id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
     controller_option VARCHAR(16),
     PRIMARY KEY (id)
);

ALTER TABLE audio_controller
     ADD COLUMN controller_v TINYINT UNSIGNED,
     ADD FOREIGN KEY (controller_v) REFERENCES
v_audio_controller(id) ON UPDATE CASCADE,
     ADD CONSTRAINT uc_audio_controller_phy_vir UNIQUE
(controller, controller_v);

CREATE TABLE v_nic_hardware_type (
     id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
     device_option VARCHAR(16),
     PRIMARY KEY (id)
);

ALTER TABLE nic_hardware_type
     ADD COLUMN device_v TINYINT UNSIGNED,
     ADD FOREIGN KEY (device_v) REFERENCES
v_nic_hardware_type(id) ON UPDATE CASCADE,
     ADD CONSTRAINT uc_nic_hardware_type_phy_vir UNIQUE (device,
device_v);

CREATE TABLE v_storage_controller (
     id TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
     ctrl_type_option VARCHAR(16),
     PRIMARY KEY (id)
);

ALTER TABLE storage_controller
     ADD COLUMN ctrl_type_v TINYINT UNSIGNED,
     ADD FOREIGN KEY (ctrl_type_v) REFERENCES
v_storage_controller(id) ON UPDATE CASCADE,
     ADD CONSTRAINT uc_storage_controller_phy_vir UNIQUE
(ctrl_type, ctrl_type_v);
```

D. BASELINE DATABASE LOOKUP TABLE INSERTION CODE

```
USE baseline;

INSERT INTO v_audio_support_types VALUES
     (null, “none”),
     (null, “null”),
     (null, “dsound”),
     (null, “oss”),
     (null, “alsa”),
     (null, “pulse”),
     (null, “coreaudio”);

INSERT INTO v_audio_controller VALUES
     (null, “ac97”),
     (null, “hda”),
     (null, “sb16”);

INSERT INTO v_gpu_controller VALUES
     (null, “none”),
     (null, “vboxvga”),
     (null, “vmsvga”),
     (null, “vboxsvga”);

INSERT INTO v_nic_hardware_type VALUES
     (null, “Am79C970A”),
     (null, “Am79C973”),
     (null, “82540EM”),
     (null, “82543GC”),
     (null, “82545EM”),
     (null, “virtio”);

INSERT INTO v_storage_controller VALUES
     (null, “LSILogic”),
     (null, “LSILogicSAS”),
     (null, “BusLogic”),
     (null, “IntelAhci”),
     (null, “PIIX3”),
     (null, “PIIX4”),
     (null, “ICH6”),
     (null, “I82078”),
     (null, “USB”),
     (null, “NVMe”);

INSERT INTO dev_state VALUES
     (null, “on”),
     (null, “off”);

INSERT INTO cpu_type VALUES
     (null, “host”),
     (null, “intl 80”),
     (null, “intl 86”),
     (null, “intl 286”),
     (null, “intl 386”);

INSERT INTO firmware VALUES
     (null, “bios”),
     (null, “efi”),
     (null, “efi32”),
     (null, “efi64”);

INSERT INTO mouse_dev VALUES
     (null, “ps2”),
     (null, “usb”),
     (null, “usbtablet”),
     (null, “usbmultitouch”);

INSERT INTO keyboard_dev VALUES
     (null, “ps2”),
     (null, “usb”);

INSERT INTO boot_device VALUES
     (null, “none”),
     (null, “floppy”),
     (null, “dvd”),
     (null, “disk”),
     (null, “net”);

INSERT INTO priorities VALUES
     (null, 1),
     (null, 2),
     (null, 3),
     (null, 4);

INSERT INTO storage_type VALUES
     (null, “ide”),
     (null, “sata”),
     (null, “scsi”),
     (null, “floppy”),
     (null, “sas”),
     (null, “usb”),
     (null, “pcie”);
```

```
INSERT INTO serial_dev_num VALUES
     (null, 1),
     (null, 2),
     (null, 3),
     (null, 4);

INSERT INTO nic_dev_num VALUES
     (null, 1),
     (null, 2),
     (null, 3),
     (null, 4),
     (null, 5),
     (null, 6),
     (null, 7),
     (null, 8);

INSERT INTO nic_hardware_mode VALUES
     (null, “none”),
     (null, “null”),
     (null, “nat”),
     (null, “natnetwork”),
     (null, “bridged”),
     (null, “intnet”),
     (null, “hostonly”),
     (null, “generic”);
INSERT INTO nic_promiscuity VALUES
     (null, “deny”),
     (null, “allow-vms”),
     (null, “allow-all”);

INSERT INTO nic_boot_priorities VALUES
     (null, 0),
     (null, 1),
     (null, 2),
     (null, 3),
     (null, 4);

INSERT INTO protocols VALUES
     (null, “TCP”),
     (null, “UDP”);
```

E. VIRTUAL DATABASE LOOKUP TABLE INSERTION CODE

```
USE virtual;

INSERT INTO audio_support_type VALUES
    (null, “none”),
    (null, “null”),
    (null, “dsound”),
    (null, “oss”),
    (null, “alsa”),
    (null, “pulse”),
    (null, “coreaudio”);
```

```
INSERT INTO audio_controller VALUES
    (null, “ac97”),
    (null, “hda”),
    (null, “sb16”);

INSERT INTO dev_state VALUES
    (null, “on”),
    (null, “off”);

INSERT INTO cpu_type VALUES
    (null, “host”),
    (null, “intl 80”),
    (null, “intl 86”),
    (null, “intl 286”),
    (null, “intl 386”);

INSERT INTO firmware VALUES
    (null, “bios”),
    (null, “efi”),
    (null, “efi32”),
    (null, “efi64”);

INSERT INTO mouse_dev VALUES
    (null, “ps2”),
    (null, “usb”),
    (null, “usbtablet”),
    (null, “usbmultitouch”);

INSERT INTO keyboard_dev VALUES
    (null, “ps2”),
    (null, “usb”);

INSERT INTO boot_device VALUES
    (null, “none”),
    (null, “floppy”),
    (null, “dvd”),
    (null, “disk”),
    (null, “net”);

INSERT INTO priorities VALUES
    (null, 1),
    (null, 2),
    (null, 3),
    (null, 4);

INSERT INTO gpu_controller VALUES
    (null, “none”),
    (null, “vboxvga”),
    (null, “vmsvga”),
    (null, “vboxsvga”);

INSERT INTO storage_type VALUES
    (null, “ide”),
    (null, “sata”),
    (null, “scsi”),
    (null, “floppy”),
    (null, “sas”),
    (null, “usb”),
    (null, “pcie”);

INSERT INTO storage_controller VALUES
    (null, “LSILogic”),
    (null, “LSILogicSAS”),
    (null, “BusLogic”),
    (null, “IntelAhci”),
    (null, “PIIX3”),
    (null, “PIIX4”),
    (null, “ICH6”),
    (null, “I82078”),
    (null, “USB”),
    (null, “NVMe”);

INSERT INTO serial_dev_num VALUES
    (null, 1),
    (null, 2),
    (null, 3),
    (null, 4);

INSERT INTO nic_dev_num VALUES
    (null, 1),
    (null, 2),
    (null, 3),
    (null, 4),
    (null, 5),
    (null, 6),
    (null, 7),
    (null, 8);

INSERT INTO nic_hardware_type VALUES
    (null, “Am79C970A”),
    (null, “Am79C973”),
    (null, “82540EM”),
    (null, “82543GC”),
    (null, “82545EM”),
    (null, “virtio”);

INSERT INTO nic_hardware_mode VALUES
    (null, “none”),
    (null, “null”),
    (null, “nat”),
    (null, “natnetwork”),
    (null, “bridged”),
    (null, “intnet”),
    (null, “hostonly”),
    (null, “generic”);

INSERT INTO nic_promiscuity VALUES
    (null, “deny”),
    (null, “allow-vms”),
    (null, “allow-all”);
```

```
INSERT INTO nic_boot_priorities VALUES
    (null, 0),
    (null, 1),
    (null, 2),
    (null, 3),
    (null, 4);

INSERT INTO protocols VALUES
    (null, “TCP”),
    (null, “UDP”);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. SQL DEVELOPER SCHEMA ANALYSIS

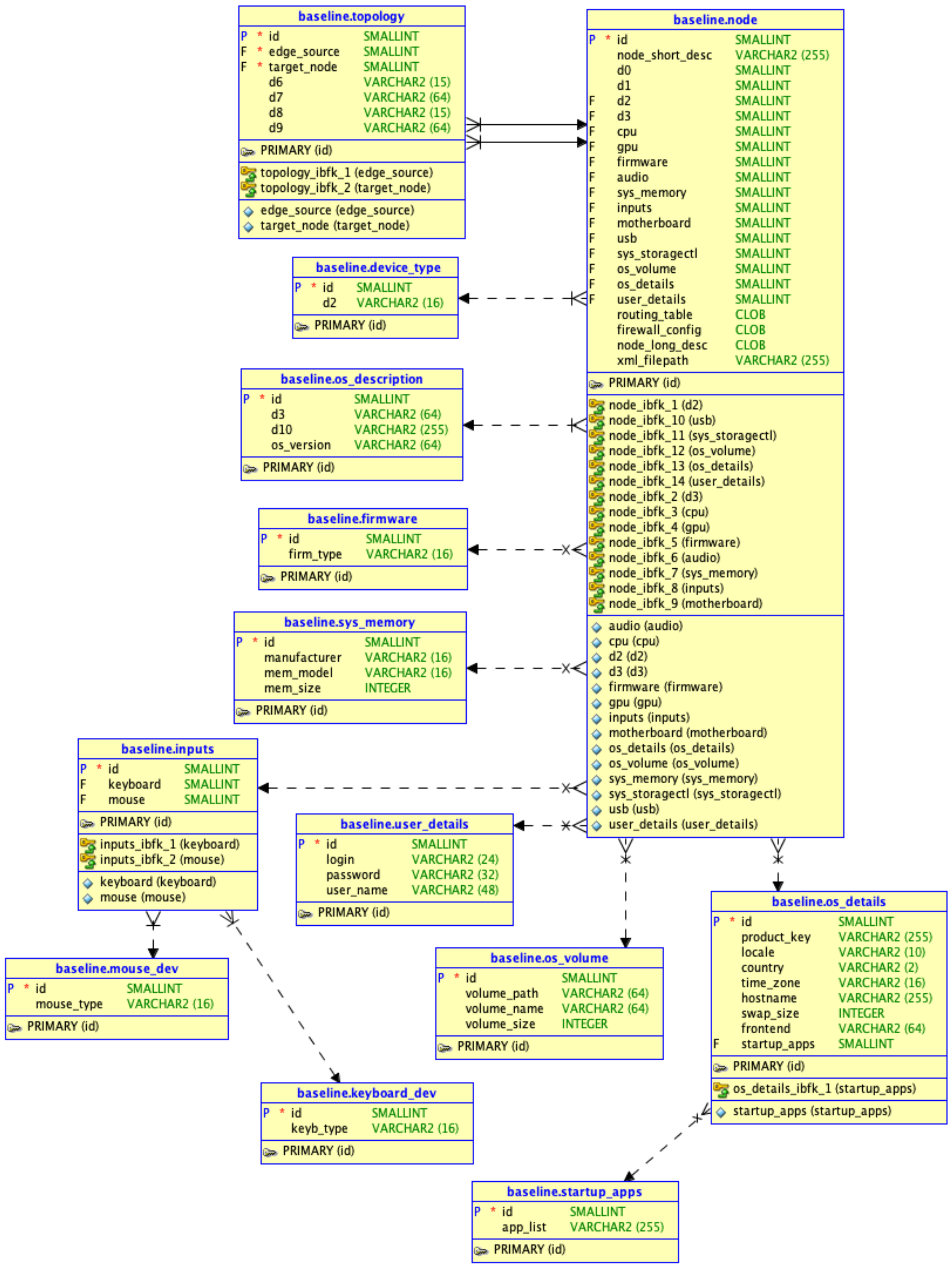


Figure 13. SQL Developer Code Validation—View 1 of 4.

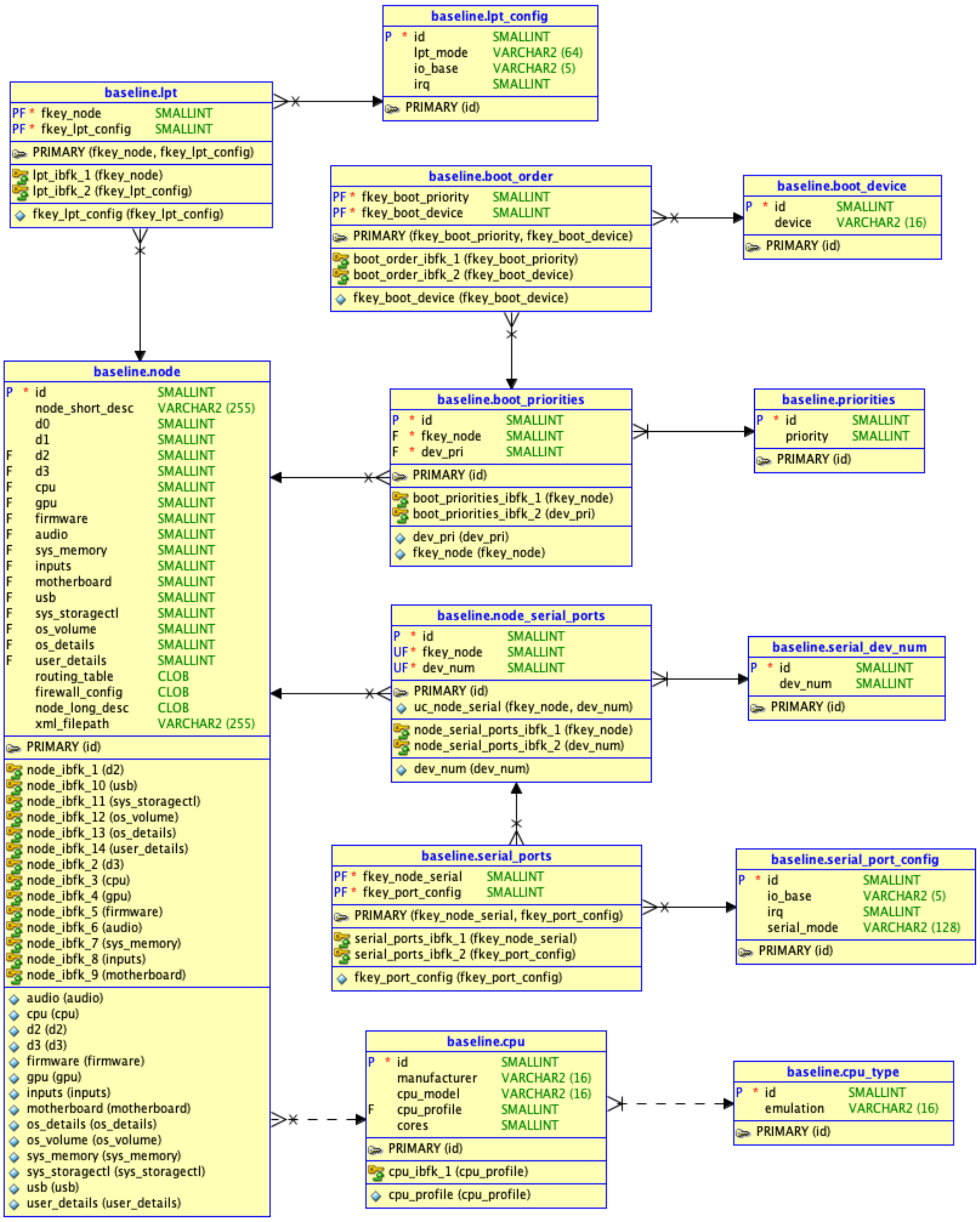


Figure 14. SQL Developer Code Validation—View 2 of 4.

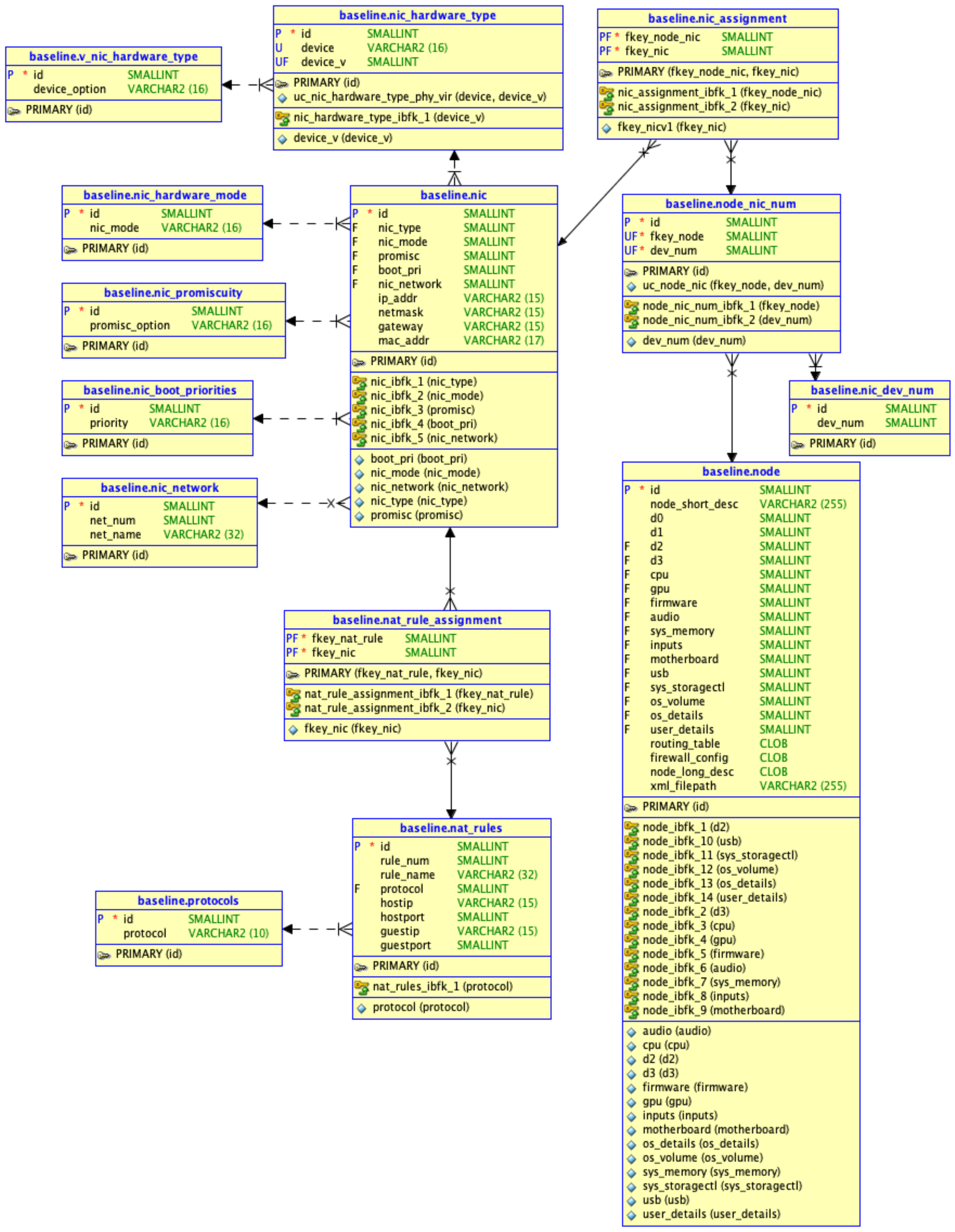


Figure 15. SQL Developer Code Validation—View 3 of 4.

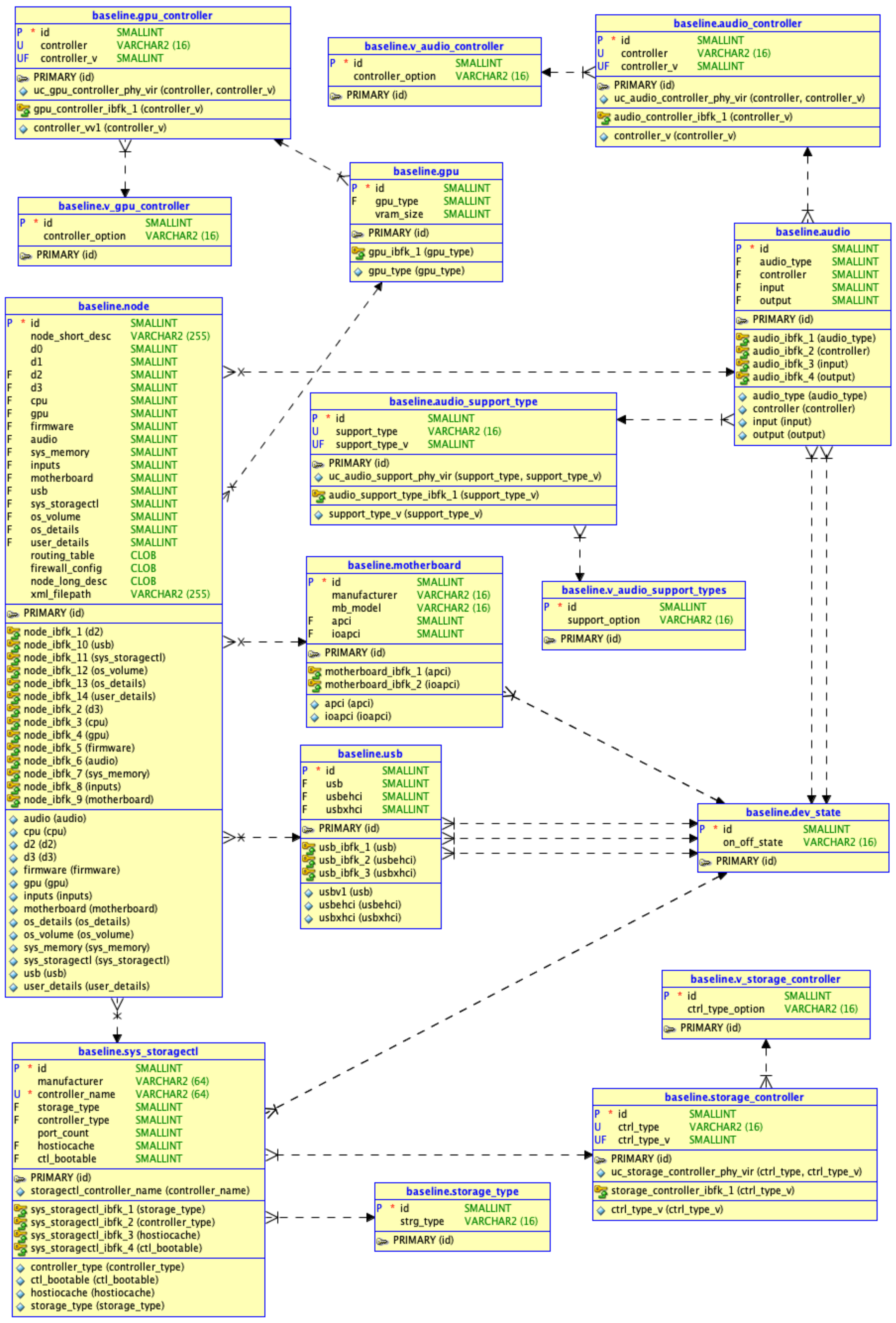


Figure 16. SQL Developer Code Validation—View 4 of 4.

LIST OF REFERENCES

Afzaal, M., Sarno, C. D., Dantonio, S., & Romano, L. (2012). An intrusion and fault tolerant forensic storage for a SIEM System. *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*, 579–586. https://doi.org/10.1109/SITIS.2012.89

Aguirre, I., & Alonso, S. (2012). Improving the automation of security information management: A collaborative approach. *IEEE Security Privacy*, *10*(1), 55–59. https://doi.org/10.1109/MSP.2011.153

AlienVault. (n.d.). About Open Threat Exchange (OTX). Retrieved January 23, 2018, from https://www.alienvault.com/documentation/otx/about-otx.htm

AlienVault. (2017). *Comparing OSSIM to USM*. Retrieved from https://www.alienvault.com/docs/whitepapers/comparing-ossim-to-usm.pdf

AlienVault. (2018). *AlienVault USM Appliance data sheet*. Retrieved from https://www.alienvault.com/docs/data-sheets/AV-USM.pdf

Anderson, S. (2014, December). CHIPS articles: A technology leap for the Navy, better quality of life for sailors. Retrieved from https://www.doncio.navy.mil/CHIPS/ArticleDetails.aspx?ID=5472

Apache Software Foundation. (n.d.). Apache NiFi Overview. Retrieved February 3, 2019, from https://nifi.apache.org/docs/nifi-docs/html/overview.html

Berndt, E. W. (2016). *Mapping, awareness, and virtualization network administrator training tool virtualization module* (Master's thesis, Naval Postgraduate School). Retrieved from https://apps.dtic.mil/dtic/tr/fulltext/u2/1027171.pdf

Bhatt, S., Manadhata, P. K., & Zomlot, L. (2014). The operational role of security information and event management systems. *IEEE Security & Privacy*, *12*(5), 5–41.

Biederman, E. (2013). *ip-netns(8) - Linux Manual page*. Retrieved from http://man7.org/linux/man-pages/man8/ip-netns.8.html

Brasetvik, A. (2013, September 16). Elasticsearch from the bottom up, part 1 [Blog post]. Retrieved February 6, 2019, from https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up

Byfield, B. (2017, March 14). 7 reasons to use Debian (and 3 reasons not to) [Online newsletter]. Retrieved January 21, 2019, from https://www.datamation.com/open-source/7-reasons-to-use-debian-and-3-reasons-not-to.html

Collier, A. R. (2016). *Automated network mapping and topology verification* (Master's thesis, Naval Postgraduate School). Retrieved from https://apps.dtic.mil/dtic/tr/fulltext/u2/1026288.pdf

Cyber. (n.d.). In *Merriam-Webster's online dictionary*. Retrieved January 6, 2018, from https://www.merriam-webster.com/dictionary/cyber

DeZyre. (2015, July 21). Spark vs Hadoop vs Storm. Retrieved February 3, 2019 https://www.dezyre.com/article/spark-vs-hadoop-vs-storm/145

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*, *37*(1), 32–64.

Franke, U., & Brynielsson, J. (2014). Cyber situational awareness–a systematic review of the literature. *Computers & Security*, *46*, 18–31.

Geil, K. (2017, September 22). *OSSIM: CIS critical security controls assessment in a Windows environment*. Retrieved from https://www.sans.org/reading-room/whitepapers/logging/ossim-cis-critical-security-controls-assessment-windows-environment-38045

Giacobe, N. (2010). Application of the JDL data fusion process model for cyber security. In *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2010*, (Vol. 7710, p. 77100R). International Society for Optics and Photonics. https://doi.org/10.1117/12.850275

Holik, F., Horalek, J., Neradova, S., Zitta, S., & Marik, O. (2015). The deployment of security information and event management in cloud infrastructure. *2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA)*, 399–404. https://doi.org/10.1109/RADIOELEK.2015.7128982

Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D. (2012). A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, *1*(6), 1-5, 2278-0181. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.678.9352&rep=rep1&type=pdf

Karaarslan, H. (2017). *A cyber situational awareness model for network administrators* (Master's thesis, Naval Postgraduate School). Retrieved from https://apps.dtic.mil/dtic/tr/fulltext/u2/1045890.pdf

Kavanagh, K. M., & Bussa, T. (2017). Magic quadrant for security information and event management. *Gartner Research Note G00315428*. Retrieved from https://www.bwdigitronik.ch/application/files/2715/1271/9597/SIEM_Magic_Quadrant_Dez_2017.pdf kcoe. (2017a, July 17). Log management in OSSIM [Online forum comment]. Retrieved January 22, 2018, from AlienVault Product Forums website: https://www.alienvault.com/forums/discussion/10642/log-management-in-ossim

kcoe. (2017b, August 22). Minimum hardware requirements for OSSIM deployment [Online forum comment]. Retrieved January 22, 2018, from AlienVault Product Forums website: https://www.alienvault.com/forums/discussion/11009/minimum-hardware-requirment-for-ossim-depolyment

Kononenko, O., Baysal, O., Holmes, R., & Godfrey, M. W. (2014). Mining modern repositories with elasticsearch. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, 328–331. https://doi.org/10.1145/2597073.2597091

Kuchera, B. (2011, August 9). Accuracy takes power: One man's 3GHz quest to build a perfect SNES emulator. Retrieved February 20, 2019, from https://arstechnica.com/gaming/news/2011/08/accuracy-takes-power-one-mans-3ghz-quest-to-build-a-perfect-snes-emulator.ars

Leszczyna, R., & Wróbel, M. R. (2015). Evaluation of open source SIEM for situation awareness platform in the smart grid environment. *Factory Communication Systems (WFCS), 2015 IEEE World Conference On*, 1–4. https://doi.org/10.1109/WFCS.2015.7160577

Lkhamsuren, T. (2012, April 25). AlienVault OSSIM review - open source SIEM. Retrieved January 22, 2018, from http://resources.infosecinstitute.com/alienvault-ossim-review-open-source-siem/

Madrid, J. M., Munera, L. E., Montoya, C. A., Osorio, J. D., Cardenas, L. E., Bedoya, R., & Latorre, C. (2009). Functionality, reliability and adaptability improvements to the OSSIM information security console. *2009 IEEE Latin-American Conference on Communications*, 1–6. https://doi.org/10.1109/LATINCOM.2009.5305052

McBride, D. C. (2015). *Mapping, awareness, and virtualization network administrator training tool (MAVNATT) architecture and framework* (Master's thesis, Naval Postgraduate School). Retrieved from http://hdl.handle.net/10945/45900

Nicolett, M., & Kavanagh, K. M. (2011). Magic quadrant for security information and event management. *Gartner Research Note G00212454*. Retrieved from http://jameskaskade.com/wp-content/uploads/2011/06/Magic-Quadrant-for-Security-Information-and-Event-Management.pdf

Oracle. (2019). MySQL document store. Retrieved January 27, 2019, from https://www.mysql.com/products/enterprise/document_store.html

OSSEC. (n.d.). Log monitoring/analysis — OSSEC 2.8.1 documentation. Retrieved October 8, 2018, from https://ossec-docs.readthedocs.io/en/latest/manual/monitoring/

Rock, C., & Bycroft, J. (2018, May 14). *SIEMonster version 3 high level design*. Retrieved from https://dyzz9obi78pm5.cloudfront.net/app/image/id/5af953a3ad121c9c30841d43/n/siemonster-v3-high-level-design-v15.pdf

Soni, M. (2018, February 19). Everything you need to know about Apache Storm [Blog post]. Retrieved February 2, 2019, from https://www.upgrad.com/blog/everything-you-need-to-know-about-apache-storm/

Sookocheff, K. (2015, September 25). Kafka in a nutshell. Retrieved February 2, 2019, from https://sookocheff.com/post/kafka/kafka-in-a-nutshell/

Sriraman, A. (2017, January 10). Docker namespace container networking tutorial [Blog post]. Retrieved January 27, 2019, from https://platform9.com/blog/container-namespaces-deep-dive-container-networking/

Tadda, G. P., & Salerno, J. S. (2010). Overview of cyber situation awareness. In S. Jajodia, P. Liu, V. Swarup, & C. Wang (Eds.), *Cyber Situational Awareness* (pp. 15–35). Retrieved from https://doi.org/10.1007/978-1-4419-0140-8_2

Turnbull, J. (2013). *The Logstash book*. Retrieved from https://books.google.com/books?id=lhMKBAAAQBAJ

Vetticaden, G. (2016, April 5). Apache Metron explained! - Hortonworks [Blog post]. Retrieved February 2, 2019, from https://community.hortonworks.com/articles/26050/apache-metron-explained.html

Vetticaden, G. (2017, July 12). Installation - Metron - Apache Software Foundation. Retrieved February 3, 2019, from https://cwiki.apache.org/confluence/display/METRON/Installation

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California