TOWARDS A UNIFIED MODELING LANGUAGE (UML) PROFILE TO ADDRESS DIGITAL FORENSIC EVIDENCE COMPLEXITIES

Robert J. Pallack Jr.
University of Rhode Island, pallack@charter.net

Follow this and additional works at: https://digitalcommons.uri.edu/oa_diss

Recommended Citation
https://digitalcommons.uri.edu/oa_diss/853

This Dissertation is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Dissertations by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons@etal.uri.edu.
TOWARDS A UNIFIED MODELING LANGUAGE (UML) PROFILE TO ADDRESS DIGITAL FORENSIC EVIDENCE COMPLEXITIES

BY

ROBERT J. PALLACK JR.

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2019
DOCTOR OF PHILOSOPHY DISSERTATION

OF

ROBERT J. PALLACK JR.

APPROVED:

Dissertation Committee:

Major Professor       Victor Fay-Wolfe
                      Joan Peckham
                      Haibo He

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2019
ABSTRACT

There is significant complexity in digital forensics due to the numerous device types and device implementations. This complexity is exacerbated by the need for digital evidence to be understood by a wide variety of stakeholders with varying technical backgrounds.

This study showed the utility of using software engineering Unified Modeling Language (UML) modeling techniques for addressing this complexity. Extensible, executable models for the digital forensics domain were developed depicting the relevant computational mechanisms involved in the who, what, when, where and how attributes of digital evidence creation.

Artifacts generated from the executable models enable a systematic constructive methodology utilizing the principle of abstraction and pattern discovery to provide a top-down view of the commonalities across implementations. It was demonstrated that the abstracted, top-down view was equivalent to implementation specific detailed views. In addition, it was shown that the executable model artifacts could be used by software applications to illustrate the creation of digital forensic evidence at various levels of detail.

Lastly, a profile was constructed to extend UML with digital forensic domain relevant concepts and vocabulary to help enable forensic domain stakeholders, who may not have a software engineering background, to apply modeling to digital forensics. The UML profile and the defined constructive methodology provided concrete artifacts to assist others in the future to develop digital forensic models.
ACKNOWLEDGMENTS

I would like to especially thank Dr. Victor Fay-Wolfe for his help, guidance, and patience throughout making my research possible. He helped me refine a very rough idea to something that was realizable. I truly am appreciative of his support and guidance throughout the process. I also am thankful to Dr. Joan Peckham and Dr. Haibo He for being part of my committee.

I am very appreciative to Jacob Fonesca and Daniel Dickerman for the review of my animation application. Their comments were an important part of the evaluation. Also, I thank Dr. Scott Fellows in his editorial comments.

Lastly, I thank my family for their encouragement. I especially thank my wife, Mary Anne, for her endless support throughout my journey.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................. ii

ACKNOWLEDGMENTS ............................................................................................... iii

TABLE OF CONTENTS ............................................................................................... iv

LIST OF TABLES .......................................................................................................... vi

LIST OF FIGURES ....................................................................................................... viii

CHAPTER 1 .................................................................................................................. 1

1 INTRODUCTION ..................................................................................................... 1

1.1 Overview forensics ............................................................................................... 2

1.2 Overview of software engineering models .......................................................... 5

1.3 Problem statement ............................................................................................... 6

1.4 Research questions and objectives ...................................................................... 8

CHAPTER 2 .................................................................................................................. 12

2 REVIEW OF THE LITERATURE ............................................................................ 12

CHAPTER 3 .................................................................................................................. 19

3 METHODOLOGY ..................................................................................................... 19

3.1 Mapping to the digital forensic problem space ..................................................... 21

3.2 Eight-Step constructive method .......................................................................... 23

3.3 Model specification: use-cases and use-case descriptions-(step 1) ..................... 33

3.3.1 File system forensics ....................................................................................... 34

3.3.2 Browser forensics ........................................................................................... 41

3.3.3 Random Access Memory (RAM) forensics .................................................... 44

3.4 Model implementation (steps 2 & 3) .................................................................. 50

3.4.1 Modeling overview ........................................................................................ 50

3.4.2 UML diagrams .............................................................................................. 51

3.4.3 Profile extensions for domain representation ................................................. 59
LIST OF TABLES

TABLE ------------------------- PAGE

Table 1-1. Stakeholders Roles ................................................................. 3
Table 3-1. Validating the Objectives ............................................................. 19
Table 3-2. Implementations ........................................................................... 23
Table 3-3 Types of Transformation Rules ....................................................... 32
Table 3-4. Browser Configurations. Adapted from [55] [56] [59]. ...................... 43
Table 3-5. Meta Levels. Adapted from [13] ..................................................... 59
Table 3-6. Related Metric Types...................................................................... 77
Table 4-1. Discovered Forensic Static Patterns .............................................. 85
Table 4-2. Dynamic Pattern Cases ................................................................ 96
Table 4-3. File System Use-Case Extension ................................................... 97
Table 4-4. Key Forensic Data Structures Mapped to Functional Groups .......... 97
Table 4-5. Browser Use-Case Extension ......................................................... 99
Table 4-6. RAM Use-Case Extension .............................................................. 100
Table 4-7. File System Functional Group/Components to Patterns ............... 102
Table 4-8. File System States ........................................................................ 105
Table 4-9. File System Profile Definition ...................................................... 108
Table 4-10. Browser Functional Group/Components to Patterns ................. 112
Table 4-11. Browser States ............................................................................ 115
Table 4-12. Browser Top-level to Profile ....................................................... 118
Table 4-13. RAM Functional Group/Components to Patterns ...................... 123
Table 4-14. RAM OS States ........................................................................... 126
Table 4-15. RAM Application States .............................................................. 127
Table 4-16. RAM Top-level Model To Profile .................................................. 128
Table 4-17. File System Equivalence ............................................................... 133
Table 4-18. DF Profile Package Description .................................................. 135
Table 4-19. Expert Review Comments ........................................................... 155
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1-1</td>
<td>Implementations</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>Problem Space</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3-2</td>
<td>Constructive Method</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3-3</td>
<td>Constructive Method Concepts</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3-4</td>
<td>Detailed Steps 1 thru 7</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3-5</td>
<td>Model Equivalence Example</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3-6</td>
<td>Example of a FAT XSLT Transformation Rule for sectorsPerDataUnit</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3-7</td>
<td>File System Use-Case Diagram</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3-8</td>
<td>File System Use-Case Description</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3-9</td>
<td>FAT File Allocation/Deletion. Adapted from [55]</td>
<td>37</td>
</tr>
<tr>
<td>Figure 3-10</td>
<td>NTFS File Allocation/Deletion. Adapted from [55].</td>
<td>39</td>
</tr>
<tr>
<td>Figure 3-11</td>
<td>EXT File Allocation/Deletion. Adapted from [55].</td>
<td>40</td>
</tr>
<tr>
<td>Figure 3-12</td>
<td>Browser Use-Case Diagram</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3-13</td>
<td>Browser Use-Case Description</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3-14</td>
<td>RAM Use-case Diagram</td>
<td>46</td>
</tr>
<tr>
<td>Figure 3-15</td>
<td>RAM Use-case Description</td>
<td>46</td>
</tr>
<tr>
<td>Figure 3-16</td>
<td>RAM Architectural View</td>
<td>48</td>
</tr>
<tr>
<td>Figure 3-17</td>
<td>Process List. Adapted from [62].</td>
<td>49</td>
</tr>
<tr>
<td>Figure 3-18</td>
<td>UML Diagrams. Adapted from Visual Paradigm (2019) Retrieved from <a href="https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is.uml/">https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is.uml/</a></td>
<td>52</td>
</tr>
<tr>
<td>Figure 3-19</td>
<td>Example Class Diagram Notation</td>
<td>55</td>
</tr>
</tbody>
</table>
Figure 3-20. Example Collection Class ................................................................. 56
Figure 3-21. Example Use-Case ................................................................. 57
Figure 3-22. Example State Machine ................................................................. 58
Figure 3-23. Files System Area Profile ................................................................. 61
Figure 3-24. Model Element Types ................................................................. 63
Figure 3-25. xtUML Class Example ................................................................. 64
Figure 3-26. Nested Data Types ................................................................. 65
Figure 3-27. OAL Operation Implementation Examples ........................................ 66
Figure 3-28. State Machine Example ................................................................. 67
Figure 3-29. Example xtUML Scenario Creation ................................................ 69
Figure 3-30. Example Initial State ................................................................. 69
Figure 3-31. Example xtUML Final State ................................................................. 70
Figure 3-32. Capturing Behavior ................................................................. 71
Figure 3-33. Example xtUML Bridge ................................................................. 72
Figure 3-34. Example xtUML External Call ................................................................. 72
Figure 3-35. Snippet to Create XML File ................................................................. 73
Figure 3-36. Sample XML Snippet ................................................................. 73
Figure 3-37. TLAM Illustrated ................................................................. 76
Figure 3-38. ICM Illustrated ................................................................. 78
Figure 3-39. Visualization Architecture ................................................................. 82
Figure 4-1. Single Elements Structure ................................................................. 87
Figure 4-2. Single Element Structure Nested Data Types ........................................ 87
Figure 4-3. SUEN Example ................................................................. 88
Figure 4-4. List Structures........................................................................................................90
Figure 4-5. List Structures with Nested Data Types .......................................................... 91
Figure 4-6. Multiple List - Directory Structure...................................................................... 92
Figure 4-7. Pointers-to-lists Structure .................................................................................. 92
Figure 4-8. Detail Structure .................................................................................................. 93
Figure 4-9. Detail Pattern FAT Example ................................................................................ 94
Figure 4-10. Details OAL Snippet ....................................................................................... 94
Figure 4-11. Dynamic Patterns ............................................................................................. 95
Figure 4-12. File System Extended Use-Case ...................................................................... 98
Figure 4-13. Browser Extended Use-Case ............................................................................ 99
Figure 4-14. RAM Extended Use-Case ................................................................................ 101
Figure 4-15. File System Top-Level Model.......................................................................... 103
Figure 4-16. File System Top-level Model Data Types.......................................................... 104
Figure 4-17. File System Top-Level State Machine ............................................................... 106
Figure 4-18. File System Static Profile .................................................................................. 109
Figure 4-19. File System Profile Tag Types ........................................................................... 110
Figure 4-20. File System State Behavior Profile ................................................................... 111
Figure 4-21. Browser Top-Level Model ................................................................................ 113
Figure 4-22. Browser Top-Level Model Data Types............................................................... 114
Figure 4-23. Browser State Machine .................................................................................... 116
Figure 4-24. Browser Use-Case Profile ................................................................................ 119
Figure 4-25. Browser Tag Data Types .................................................................................. 120
Figure 4-26. Browser State Behavior Profile ....................................................................... 121
Figure 4-27. RAM Top-Level Model ................................................................. 125
Figure 4-28. RAM Top-level Model Data Types............................................. 126
Figure 4-29. RAM OS State Machine................................................................. 126
Figure 4-30. RAM Application State Machine ................................................. 127
Figure 4-31. RAM Static Profile........................................................................ 129
Figure 4-32. RAM Profile Tag Data Types ......................................................... 130
Figure 4-33. RAM OS State Profile................................................................. 131
Figure 4-34. RAM Application State Profile .................................................... 131
Figure 4-35. DF Profile Package View ............................................................ 134
Figure 4-36. DF OS Profile ............................................................................. 137
Figure 4-37. DF Application Profile ............................................................... 138
Figure 4-38. DF File System Profile ............................................................... 139
Figure 4-39. DF RAM Profile ......................................................................... 140
Figure 4-40. DF Evidence Types ................................................................. 141
Figure 4-41. DF Types .................................................................................. 142
Figure 4-42. TLAM Results .......................................................................... 144
Figure 4-43. ICM Results ............................................................................ 146
Figure 4-44. ICM File System Results ........................................................... 146
Figure 4-45. ICM Case 3 (Across Use-Case) Results ...................................... 147
Figure 4-46. Constructive Method ................................................................. 148
Figure 4-47. Application View (No Details) .................................................... 152
Figure 4-48. Application View (Detailed) ........................................................ 153
CHAPTER 1

1 INTRODUCTION

There is significant complexity in digital forensics. Digital forensic analysis is performed on devices which are complicated systems that contain a large number of components between which can exist a practically uncountable number of interactions. Devices can be composed of different component types and component implementations. In addition, there could be multiple devices involved in a digital forensic scenario. This complexity is exacerbated by the fact that digital evidence needs to be understood by a wide variety of stakeholders with varying technical backgrounds.

This problem space is analogous to the development and maintenance of software systems. Software systems can be incredibly complex, consisting of software applications and components that can have multiple implementations and practically uncountable interactions. Like forensic digital devices, software systems also have stakeholders with varying technical backgrounds. Thus, the question becomes: can software engineering approaches be applied to digital forensics to assist in managing digital forensic complexities?

In fact, most forensic devices and components are implemented by software. So, it was reasonable to explore whether modeling techniques used on software systems would apply to digital forensics.

Models can facilitate the development of views, techniques, and tools, which can enhance the understandability of evidence in computational devices. Tools implemented as software applications can utilize models to abstract relevant
information to support multiple stakeholder viewpoints. Models can be extended to address both existing and future implementations of the devices and components that comprise the systems on which digital forensics are performed. Models can facilitate the discovery of common implementation patterns across different device implementations. These patterns can be utilized as a means of simplifying domain complexities that are reflected in abstracted top-level models. The identification of patterns also facilitates the development of domain-specific model profiles to describe the system being modeled in the terminology of the domain stakeholders. The model profile can then be utilized to support future model development by the domain stakeholder.

The top-level model needs to evolve over time to address additional implementations as new devices or versions of devices are introduced. The profile also needs to be extended to address additional forensic use-cases, as required.

This chapter introduces forensic complexities and software engineering modeling. The problem statement is documented through a discussion of the ways in which modeling can address digital modeling complexities. Lastly, research questions, the hypothesis, and objectives are identified.

1.1 Overview forensics
In digital forensics, stakeholders are concerned with understanding the who, what, where, when, and how attributes of digital evidence. They need to know what evidence is available and where to look for it. In addition, they need to know, from a timeline perspective, when the evidence was created and, if possible, who created it.
Also, to defend the validity of the digital evidence, it is important to know how the evidence was created and how it could be changed.

To be able to answer these questions, the stakeholders need to have an understanding of how digital evidence is created on a device to the appropriate level of abstraction (e.g., detail) for their role (see Table 1-1). To do this, they need to understand the computational mechanisms, the device’s relevant underlying component data structures and the operations performed on these data structures. For example, an analyst needs a deep understanding of the underlying computational mechanisms of the device to address all aspects of the evidence and to ensure that their conclusions are defensible. A witness needs to understand how the evidence relates to a case, but only needs to understand the computational mechanisms of the device to the level that they have confidence in the validity of the evidence.

Table 1-1. Stakeholders Roles

<table>
<thead>
<tr>
<th>Stakeholder Role</th>
<th>Level of Understanding Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Forensic Analysis</td>
<td>Requires expert level understanding of digital forensics to determine what information is available and how to recover the information.</td>
</tr>
<tr>
<td>Lawyer/forensic expert testimony</td>
<td>Strong level of understanding to convey important concepts to a lay audience and to defend analyst results.</td>
</tr>
<tr>
<td>Juror</td>
<td>Understand the concepts to aide decision making. May have minimal technical background.</td>
</tr>
<tr>
<td>Educator</td>
<td>Strong level of understanding of the concepts to teach others.</td>
</tr>
<tr>
<td>Student</td>
<td>Learning concepts. Depending on context, may or may not have strong technical expertise.</td>
</tr>
<tr>
<td>Law Enforcement</td>
<td>Understand concepts to ensure complete and reliable collection of data. May have minimal technical background.</td>
</tr>
</tbody>
</table>

Digital forensic investigations tend to be complex in that they can include
numerous computational device types implemented by multiple components, with each component potentially having multiple implementations (see Figure 1-1). In the context of this dissertation, a device is a computer-based system (e.g., computers, tablets, phones, etc.). Devices are composed of components (e.g., applications, operating systems, file systems, network connections, etc.). Components may be implemented in numerous ways (e.g., operating systems (OS) can be Windows, Linux, MAC, etc.; files systems can be FAT, NTFS, EXT; etc.).

There is additional complexity in that a device is not static, but rather has dynamic behavior that is significant in understanding the creation of evidence. Devices interact with users and other devices, and components within a device interact with other components. This research focused on the modeling of devices, device components, and the interactions of the device components.
1.2 Overview of software engineering models

Software models capture both the static and dynamic (e.g., behavioral) design aspects of a system. A formal model is computationally rigorous and can be directly utilized in both the development and runtime aspects of applications. A model provides an abstraction of the system.

Software modeling has been used for several decades to address software complexity. The Unified Modeling Language (UML) is the graphical modeling language that is widely accepted in the software community. UML provides a means to model both static and dynamic behaviors of the system. UML can address both software and hardware in systems.

UML provides a formalism such that the model’s graphical depiction is consistent with an underlying mathematical basis. UML-based models can be constrained so that they are unambiguous, which allows models to be internally consistent and executable as a programming language. Through model execution, the model behavior can be observed and recorded.

UML modeling utilizes principles, such as abstraction, to address complexity. Abstraction ensures that only the important details necessary for a particular stakeholder are addressed. In addition, UML can also be used to identify common structural and behavior patterns of the system being modeled. This is beneficial since one representation of an implementation is less complex than having multiple unique representations of the same underlying device or component functionality.

UML utilizes object oriented terminology and software engineering concepts, which can be a barrier for individuals without a software background. To address this, a profile can be created to extend UML to address a domain (i.e., a specified sphere of
activity or knowledge). The profile allows for a system to be modeled with concepts and terminology familiar to a domain stakeholder. As an example, a widely accepted UML profile for system engineering is the System Modeling Language (SYSML). SYSML is used by numerous industries (e.g., auto, railway, defense, etc.). This work explored using UML profiles for the digital forensic domain.

1.3 Problem statement

There is significant complexity in digital forensics due the numerous possible device types and device implementations. This complexity is exacerbated by the need for digital evidence to be understood by a wide variety of stakeholders with varying technical backgrounds.

**How does modeling address the problem?** Software modeling techniques can help address the problem of digital forensic complexity by providing an understanding of the underlying computational mechanisms involved in the creation of evidence, providing a top-level view that shows that which is common across implementations and facilitating the development of applications to assist in the understanding of digital forensic concepts.

Models can be used as a means to formally document the key computational mechanisms that are involved in digital evidence creation. Models can be shared and evolved by digital forensic practitioners. Since the model requires specific formal notations, it allows for non-ambiguous descriptions. The model can also become an authenticated source to the community. Models need to be extensible so that they can change overtime. For example the model may need to increase the level of abstraction
(i.e., model fidelity) for an analyst stakeholder investigating an aspect of a computational mechanism. Models are extensible and will be able to address a new implementations, new forensic categories, new devices, and new evidence creation use-cases.

Using abstraction to reduce complexity is a concept that is commonly used in software engineering. Since digital forensics is accomplished on software-based systems, it is reasonable to assume that this concept is applicable to the digital forensics domain to manage the complexity of the combinatorically explosive number of implementation configurations. The premise behind the use of abstraction is that it is often easier to understand a general concept than to understand all the implementation details. In addition, common structural and behavioral patterns can be identified from implementation specific models and used to construct an abstract top-level model. A top-level model can provide one representation for several implementations by identifying that which is common across unique implementations and abstracting out the details.

Applications are regularly available in many fields and domains as a learning mechanisms. Models can be utilized to facilitate the development and runtime aspects of applications. Making learning applications more available in the digital forensic field can only help in facilitating understandability.

Top-level models and digital forensic learning applications can both represent different levels of abstraction. This potentially allows for the needs of all potential digital forensic stakeholder to be addressed. Lastly, profiles can more readily make software modeling techniques available to digital forensic practitioners.
1.4 Research questions and objectives

The questions to be answered in this study were:

1. Can it be shown that there are commonalities in the device and component implementations on which digital forensics are based?

2. Is there utility in applying software engineering modeling techniques to digital forensics in terms of managing complexity and in promoting understandability?

3. If so, is there a potential approach in applying software modeling holistically across the digital forensic domain?

The first question was addressed in this work by answering the following question:

- Can implementation commonalty be identified and measured?

The second question was addressed in this work by answering these questions:

- Can models be used to create top-level generalized diagrams?
- Can models facilitate digital forensic learning applications?

The third question was addressed in this work by answering these questions:

- How can models be holistically applied to the digital forensics domain?
- Can modeling be more accessible to digital forensic stakeholders to reduce the need of software engineering skill sets?

This work hypothesized that there are commonalities in the implementation of the components and devices on which digital forensics are performed and that this
implementation commonality along with software engineering modeling approaches can be applied to address digital forensic complexities. To address the hypothesis, the four specific objectives of this study were:

- **Objective 1:** Address domain complexity by utilizing model artifacts to facilitate the development of applications for learning and analysis that support multiple digital forensic stakeholder roles.
- **Objective 2:** Address domain complexity by providing a top-down view and to find commonality patterns across implementations.
- **Objective 3:** Show the extensibility of models to address the introduction of different implementations and different implementation types.
- **Objective 4:** Construct a digital forensic domain specific UML Profile for digital evidence creation that can be used for future modeling efforts.

1.5 Contributions of this work

This work identified a unique approach to extend software modeling into the digital forensic domain by focusing on digital forensic complexities. Numerous contributions resulted:

1. Modeled the computational mechanisms involved in the creation of digital forensic evidence,
2. Generated computable model artifacts to facilitate the execution of a digital forensic relevant animation application,
3. Identified a process to construct top-level implementation views of computational mechanisms,
4. Identified modeling patterns to catalog commonalties in computational mechanisms,

5. Identified metrics to support commonality and abstraction analysis, and

6. Introduced a digital forensic profile along with a process to extend the profile.

By modeling the underlying computational mechanisms, insight and understanding can be gained on how evidence is created, which would be beneficial to domain stakeholders. The computational mechanisms were both statically and behaviorally modeled, and as a result, the model could be executed and model behavior captured. The captured model behavior can be parsed which allows algorithms to be written to utilize the captured behavior to facilitate digital forensic applications. In the case of this work, the captured model artifacts were utilized to generate a script for an animation application. The animation application could be used as a teaching aide.

Digital forensics tends to be a bottom-up process in that evidence gathering procedures focus on specific implementations. This work introduces a top-down approach which utilizes commonalities across implementations. The top-level models and associated top-down views, are more abstract than the implementation models and will provide alternate approaches in addressing digital forensics complexities.

Patterns were utilized to identify commonalities across the computational mechanisms being modeled. The identification of patterns was utilized to determine the degree of commonality in the different implementations of the computational mechanisms being modeled. In addition, the patterns can be reused in other modeling efforts and were also utilized to construct this works top-level models.
Commonality and abstraction measures developed for this work provide a means of quantifying commonality between models and levels of abstraction between models. These metrics were utilized to assess potential relationships between these model and implementation properties.

Lastly, this work identified an initial digital forensic for the computational mechanisms which are the subject of this study. The profile identifies a set of modeling elements which define the model elements which are relevant to the vocabulary and concepts of the digital forensic domain. A process on how to extend the profile was also defined.
CHAPTER 2

2 REVIEW OF THE LITERATURE

The validity of digital forensic data can be subject to significant scrutiny, such as the highly publicized Anthony murder case in which the digital forensics tools provided contradictory results [1] [2]. There is a critical need to train computer forensic professionals to properly gather all relevant evidence and to have the staff to process evidence in a timely manner [3]. The most significant challenge in digital forensics is the lack of qualified people and recommended the development of new tools and capabilities [4].

Visualization techniques can be used to enhance learning and understanding. Visualization has been used in addressing information that might be of interest in forensic investigations [5]. Visualization has been also specifically used for understanding of digital forensic information [6] [7] [8] [9] [10] [11].

UML is often used to facilitate application development as illustrated in [12]. Digital forensics investigations are performed on software-based devices that utilize standard computer architectures. Therefore, software engineering modeling techniques are also available to the digital forensics domain.

Models manage complexity by formally capturing both the static and dynamic design aspects of a system. A formal model is computationally rigorous and can be directly utilized in both the development and runtime aspects of applications. A model provides an abstraction of the system. Abstraction removes detail to allow a higher level view through which to facilitate understanding [13]. Model abstraction needs to provide the level of detail required to address the generalized attributes that are of interest, in this case the digital evidence attribute.
Patterns in software engineering are utilized to make designs more efficient by reusing common design approaches [14]. Patterns are documented with artifacts such as UML class, object, and sequence diagrams. An approach to specify UML patterns was discussed in [15]. Additionally, [16] discussed pattern types and how patterns can be utilized in developing domain-specific models.

UML is a widely accepted software modeling approach and is an Object Management Group (OMG) standard [17]. UML can be used to model both static and dynamic aspects of software. Model Driven Architecture (MDA) utilizes UML to support design by providing capabilities such as model execution and model transformation (e.g., code generation). Models can be executed, similar to code, on model virtual machines to simulate model behaviors utilizing frameworks that are based on the OMG Foundational UML [18] and OMG Action Language to Foundational UML (ALF) standard [19] [20]. An example of an equivalent modeling framework implementation is eXecutable Translatable UML [21]. Executable models are of significant importance to this research since they allow the actual behaviors of the modeled device/device components to be captured and utilized by applications.

UML profiles are a mechanism for extending UML to reflect the terminology and concepts of a particular domain. UML profiles provide a concise dialect that consists of stereotypes (i.e., new model meta-elements), tags (e.g., attributes) and constraints that are support by UML compliant tools [22]. The stereotypes and tags capture domain terminology/concepts. Object Constraint Language (OCL) rules are utilized to define the constraints, which are used to provide additional model precision and can be used for model validation. OCL is an extension of UML [23] that can
formally specify UML.

Software engineering metrics have been developed for objected oriented software systems [24]. Examples of software reuse metrics are seen in [25].

**Modeling and formalism in digital forensics and cyber security.** In addressing modeling in digital forensics, it is useful to also look at how modeling is used in cyber security. The cyber security domain is directly related to the digital forensics domain in that the underling systems to be investigated or analyzed are computer systems that are based on similar technical concepts at similar levels of abstraction. In the digital forensic use-case, the analysis is reasoning about the existence of digital forensic evidence on the computer system, whereas in the cyber security use-case, the focus is on computer system security vulnerabilities.

Although there are a number of papers devoted to the use of formalism in digital forensics and cyber security, they have a different focus than the model proposed from this work. There is a significant amount of literature recommending the use of modeling in digital forensics to formally model the digital forensic process [26] [27] [28] [29] [30].

The formalism of modeling can be utilized algorithmically to reason about the system being modeled. A Turing Machine-Based model to address evidence is identified in [31]. A modeling method for forensic analysis formally using graphs to address attack vulnerabilities is discussed in [32]. The use of modeling for the analysis of evidence in storage media is introduced in [33].
Garfinkel [34] defined a limited XML schema to formally capture forensic case information. The intent of the schema was to provide an Application Programmer Interface (API) for digital forensic tools to share data sets.

The cyber security modeling focus is on identifying system vulnerabilities and identifying likely attack scenarios [35] [36]. These models incorporated aspects of the underlying system architecture and in some cases also included a model of the human element. A method to extend UML to address security concerns has been introduced in [37].

Another aspect of formalism are Domain Specific Languages (DSL), tailored programming languages in which domain lexica and concepts are built into the language. DSLs containing digital forensics and cyber security constructs are to be or have been developed [38] [39] [40] [41].

Domain Specific Model Languages (DSML) is a model-based approach analogous to DSLs, but at a higher level of abstraction. DSMLs are targeted for domain stakeholders and provide an abstract generic model instead of a specific programming language implementation. Like DSL models, DSML models can also be directly executed or compiled to provide a transformation to other artifacts utilizing MDA techniques. UML profiles have a strong relationship to DSMLs and information between UML profiles and DSMLs can be interchanged [42] and profiles can be used as a mechanism to design DSMLs [43].

In the cyber security realm utilizing profiles to incorporated security patterns is discussed in [44]. The utilization of security patterns for development of more cyber resilient systems is addressed in [45] [46]. Fernandez and Petrie [47] suggested that
UML and security patterns can be used as a mechanism to teach secure system design. An example of the utilization of a pattern in digital forensics to isolate forensically interesting network data was reported by [48].

*Modeling in other domains: System Engineering Modeling Language (SysML).* A significant example of extending UML for other disciplines is seen in the System Modeling Language (SysML). SysML is a profile extension of UML with a focus on system engineering of complex systems and system-of-systems through their lifecycles. SysML has been applied to complex systems in many industries, including aircraft, automotive, defense, IT, medicine, and space systems. As an example, the utilization of SysML for auto-embedded systems are discussed in [49] [50]. The application of SysML for railroad crossings was identified in [51].

*Contributions of this study.* This study showed the benefits and provided a unique approach to extend software modeling techniques into the digital forensic domain. The actual subjects of the model were the computational mechanisms of the software architecture of evidence creation. The computational mechanisms as defined by this work were the forensically relevant data structures and control flow as dictated by the relevant device component(s). In other works for digital forensics, UML was used to model evidence acquisition, analysis of evidence, and attack methodologies. Other modeling techniques were utilized to reason on the validity of evidence. These works did not address the system level constructs of the computational mechanisms. None of
the UML modeling techniques investigated for this study addressed executable models nor utilizing the outputs of executable models to be utilized by other applications.

Other works did address utilizing UML to produce animations. However, these animations were not related to digital forensics. This work utilized the formalism which UML provides to generate formal artifacts which could be parsed and animated for a forensic application. This animation provided insight on forensic applications based on models that can enhance domain understanding.

This work utilized the principle of abstraction along with discovered commonality patterns to define top-level models from detailed implementations. A resulting repeatable modeling process on how to construct the abstracted top-level was identified. An equivalent process was not seen in the literature review.

Patterns are common in object oriented software development. Higher level patterns have been developed for security design patterns and attack patterns for cyber security. Although patterns have been addressed for cyber security, they were not addressed in any of the literature reviewed on digital forensics. The patterns developed for this work are unique in that they define what is common in the data structures and control flow of the computational mechanisms which are relevant to evidence creation.

The other works reviewed during this study identified various types of software-based metrics. This work extended the metrics of other works to introduce a new metric for addressing commonality across implementations. In addition a second new metric was developed to quantify the level of abstraction between top-level models and models of specific implementations.
Profiles are a common mechanism to extend UML in other domains, as seen with SysML. However, the literature search did not identify any digital forensic related profiles. This work also identifies a process to construct profiles from top-level views.
CHAPTER 3

3 METHODOLOGY

This work has two primary focus areas, the construction of top-level models and profiles, and the development of a learning application utilizing models artifacts. The first focus area results in the construction of the top-level models to address digital forensic complexities through abstraction. The top-level models facilitate the construction of profiles which make modeling more accessible to digital forensic stakeholders. The second focus area addressed complexities by demonstrating that modeling could facilitate the development of applications utilized to enhance the understanding of digital forensic stakeholders.

A combination of expert review, analysis, test, and metrics were used to show that the objectives were met, see Table 3-1.

Table 3-1. Validating the Objectives

<table>
<thead>
<tr>
<th>Objective</th>
<th>Success Criteria</th>
<th>Validation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Facilitate Learning and Analysis Application</td>
<td>Model based application has utility for a user role.</td>
<td>Expert Review.</td>
</tr>
<tr>
<td>1: Facilitate Learning and Analysis Application</td>
<td>Application provides model abstractions for multiple user roles.</td>
<td>Expert Review.</td>
</tr>
<tr>
<td>2: Reduced Complexity</td>
<td>Common Implementation Patterns Discovered.</td>
<td>Analysis (bottom-up) of specific models.</td>
</tr>
<tr>
<td>2: Reduce Complexity</td>
<td>Top-level models developed for a set of implementations.</td>
<td>Analysis utilizing construction.</td>
</tr>
<tr>
<td>2: Reduced Complexity</td>
<td>Top-level model equivalent to specific implementation models.</td>
<td>Test utilizing transformation.</td>
</tr>
<tr>
<td>2: Reduced Complexity</td>
<td>Increased commonality and increased abstraction shown in metrics.</td>
<td>Metrics.</td>
</tr>
<tr>
<td>3: Extensibility</td>
<td>Repeateable procedure to extend models for implementation and implementation types.</td>
<td>Analysis utilizing construction.</td>
</tr>
<tr>
<td>3: Extensibility</td>
<td>Repeateable procedure developed to extend profiles.</td>
<td>Analysis utilizing construction.</td>
</tr>
<tr>
<td>4: DF UML Profile</td>
<td>A Profile constructed from models.</td>
<td>Analysis utilizing construction</td>
</tr>
</tbody>
</table>
Constructive methods “are heuristics that build up a complete solution from
scratch by sequentially adding components to a partial solution until the solution
is complete” [52]. For this work, a constructive method was utilized to define the
top-level models and profiles. In addition, the constructive method provided the
steps to perform the constructive analysis and test, as identified in Table 3-1. The
constructive method itself was shown to be repeatable and was one of the
outcomes of this work. The constructive method provided a procedure to extend
models and profiles.

Two metrics were used in this work to determine the level of abstraction and to
assess commonality for various steps in the constructive method. The Top-Level
Abstract Metric (TLAM) characterized the reduction of model elements of the
abstracted top-level model with the elements in the implementation specific models.
The Implementation Commonality Metric (ICM) was developed to measure
commonality. The more commonality that existed resulted in fewer types of
components that needed to be taken into account, thus reducing complexity.

Lastly, an application was developed to animate the behavioral script for a
specific model. The animated application demonstrated that a model could be used
directly to generate an application, which could assist the human stakeholder in
understanding digital forensic complexities. This was an example of how models
could facilitate the development of tools/applications to increase human
understandability. The University of Rhode Island (URI) Digital Forensics and Cyber
Security Center (DFCSC) [53] staff evaluated the application.
This chapter begins by addressing the digital forensic problem space and how it relates to this work. Next, the constructive methodology is discussed, the forensic subjects to be modeled are identified, followed by the modeling implementation approach. The metrics to address commonality and complexity are then introduced. Lastly, the animated application is described.

### 3.1 Mapping to the digital forensic problem space

The digital forensic problem space that this work addressed is combinatorically explosive. There is an uncountable number of digital forensic scenarios that can be executed on practically an uncountable number of implementations of devices and device components.

Figure 3-1 displays how this work addressed the problem space. Forensic areas in the context of this work include media analysis, media management analysis, file system analysis, application analysis, network analysis, operating system analysis, executable analysis, image analysis, and video analysis, as originally identified by [54]. Additional areas that were added over time include RAM (Random Access Memory), mobile, and database forensics.

Given a forensic area, there are forensic use-cases which identify specific investigation types. A use-case, in the context of this work, was a set of actions performed by the suspect on the targeted system or device that would be of interest to the forensic stakeholder. A use-case scenario was one realization of a use-case which requires an initial configuration. A forensic attribute was either the evidence or contributes to evidence identification for the forensic stakeholder.
This work utilized use-case descriptions to describe a high-level scenario and the forensic attributes that were of interest for a particular type of forensic evidence creation. The descriptions were accompanied by an associated use-case diagram that provided additional details of the activities, the system boundary, and participants (e.g., actors) in the scenario. The combination of the use-case description and use-case diagram provided the specification for that which was modeled.

The forensic scenario and forensic attributes of what, when, where of the evidence assist in determining the data structures of the specific implementation. It should be noted that the forensic attributes may be at a much higher level of abstraction than the actual implementation of the underlying device or component. The how of evidence creation is reflected in how the underlying computational data structures are utilized and how they change during a scenario.

![Figure 3-1. Problem Space.](image)

This work focused on developing models for the application, file system, and RAM analysis areas. The application analysis area was chosen because there is an
ever-increasing number of application types and implementations. The file system area was chosen because the file system is utilized by many applications and there exist a large number of file system implementations. The RAM area was chosen because it contains key evidence not available through other types of analysis since it provides insight into the state of the operating system. RAM analysis performs forensic analysis on what resides in RAM at the time it was captured. The models created in this project were based on materials utilized in the URI digital forensic courses and published articles. The specific analysis areas, implementations, and use-cases used for this work are shown in Table 3-2.

Table 3-2. Implementations

<table>
<thead>
<tr>
<th>Analysis Areas (Browser)</th>
<th>Implementation 1</th>
<th>Implementation 2</th>
<th>Implementation 3</th>
<th>Evidence Creation Use-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application (Browser)</td>
<td>Edge</td>
<td>Safari</td>
<td>Chrome</td>
<td>Browsing history, downloads, cookie artifacts</td>
</tr>
<tr>
<td>Memory/RAM</td>
<td>Windows</td>
<td>Linux</td>
<td>Mac (e.g., macOS,iOS)</td>
<td>Backdoors and remote user access</td>
</tr>
<tr>
<td>File Systems</td>
<td>FAT</td>
<td>NTFS</td>
<td>EXT</td>
<td>File allocation/deletion</td>
</tr>
</tbody>
</table>

3.2 Eight-Step constructive method

The eight steps which comprised the constructive methodology are detailed in Figure 3-2. In the diagram, the set of steps are shown along with the focus of the step. In addition, the steps in which metrics were taken are identified. Also shown are the mapping of the steps to the objectives. The constructive method first specifies what is to be modeled. There are three use-case scenarios modeled in this work, each consisting of three implementations. This resulted in the eight-step methodology being completed three times.
Figure 3-3 identifies the significant products of the constructive methodology. The use-case description and use-case diagram specify what is to be modeled. Figure 3-4 provides a graphical representation of the types of modeling artifacts utilized for the first seven steps of the constructive method.

The specification was used to develop the implementation specific models. Once developed, the implementation specific models were analyzed for commonalities from both a black box and white box perspective. From a black box perspective, common functionality across the implementations was identified as a functional group. The functional groups were utilized to extend the initial use-case. The resultant extended use-case described the top-level model. From a white box perspective, the forensic data structures and scenario control flow were analyzed to determine common implementation patterns.

The top-level model implements the functional groups and the associated use-case scenario. An analysis was performed to determine which implementation pattern should be utilized to implement the functional group. The top-level and implementation models were shown to be equivalent by ensuring that the attributes of the modeled data structures could be transformed to the forensic attributes.

From the function top-level models, the use-case profile can be defined. The three use-case profiles constructed from this work were integrated to develop the holistic digital forensic profile.
Figure 3-2. Constructive Method.
Figure 3-3. Constructive Method Concepts
Figure 3-4 Detailed Steps 1 thru 7
**Model specification (step 1).** The first step was to determine the use-cases and use-case descriptions. The use-cases identified the actors, the systems (i.e., devices), the activities or functions, and components utilized by a suspect for a given scenario. The high-level functions were typically at the operating system or at the application level (e.g., Microsoft Word, Chrome browser, command line, etc.). After the use-case and the associated scenario were determined, the associated forensic attributes were identified. The use-case description defined a specific usage of the use-case. The forensic attributes set the model abstraction level.

**Develop specific models (step 2).** There were three specific model implementations developed for the use-case. The relevant components (e.g., operating system, applications) and forensic data structures of the implementation were the focus of the static model. The operating system and applications were the typical components in the model along with the forensic data structures. The forensic data structures were often modeled from tables in forensic documentation or documentation which described the functionality of interest. The use-case scenario was the basis for the behavioral model that was developed as a state diagram and utilized action language. In addition, the initial conditions for each implementation scenario where defined.

**Execute models (step 3).** The specific models were instrumented so that upon model execution, an XML script was created. The XML script logged the behavior of the model during the implementation with respect to the forensic attributes. In the
specific models, these captured attributes may not have been the exact forensic attributes, but they could be related to the forensic attributes. For each implementation, the model behavioral script was verified against the source documentation to ensure the model implementation exhibited the expected behavior.

**Model analysis (step 4).** The implementation of specific models were analyzed to identify common implementation patterns and functional groups. The static implementation patterns focused on the underlying forensic data structures of the implementations. The dynamic patterns focused on the control flow of the scenario. The implementation patterns provide a white box definition of the underlying computational mechanisms.

The functional groups were the grouping of the common functionality across the implementations in support of the activities defined in the use-cases. The functional groups provided a black box definition of the underlying computational mechanisms. The initial use-case diagrams were extended to incorporate the additional functionality identified by the functional groupings. Metrics were collected to identify the common patterns as opposed to the unique model elements in each implementation.

**Develop top-level model (step 5).** The top-level model was composed of the components and the functionality as defined by the functional groupings. Analysis was needed to determine which common implementation patterns were to be utilized to model the functional groups in the top-level model. In addition, a determination
needed to be made about which component provided the control thread. This component would contain the dynamic behavior that was defined by the state diagram. Associated Object Action Language (OAL) was created to achieve the desired behavior of the generation of the forensic attributes for the use-case scenario. Metrics were taken to quantify commonality across the use-case and to quantify model abstraction.

**Validate top-level model (step 6).** Model equivalence showed that the behavior of the top-level model was equivalent to the behavior of all the specific models for a given scenario. Figure 3-5 shows conceptually how model equivalence was determined. The executable models were instrumented such that an XML script was generated, capturing the behavior of the top-level model and the behavior for each implementation-specific model with respect to the forensic attributes of the given scenario.

To verify that the specific implementation models were equivalent to the top-level model, eXtensible Stylesheet Language (XSLT) transformation rules for each specific model were developed to map the specific model attributes to the forensic attributes of the top-level model. It was also verified that the top-level model behavioral script addressed the forensic attributes utilizing XSLT.

This mapping was analyzed to assess equivalence. If the top-level and specific models executed the same use-case scenario and each model either directly accounted for the forensic attributes or could map to the forensic attributes through a
transformation, it was reasonable to claim that the specific models were equivalent to the top-level model.

![Diagram of XSLT transformations](image)

**Figure 3-5. Model Equivalence Example**

An example of an XSLT transformation is shown in Figure 3-6. The XSTL rule is used to transform the specific models attribute of `clusterSize` to the top-level model’s attribute of `dataUnitSize`. Also, this attribute was calculated by the equation `bytesPerSector * sectorsPerDataUnit`.

![XSLT transformation rule example](image)

**Figure 3-6. Example of a FAT XSLT Transformation Rule for sectorsPerDataUnit**

As shown in Table 3-3, four transformation rules were identified: direct mapping, equivalent mapping, intermediate step, and calculation. A direct mapping
transformation occurred when there was no difference between the attributes of the top-level and specific model, of which `bytesPerSector` attribute is an example. An equivalent mapping transformation occurred when both specific and top-level models used the concept without a direct one-to-one mapping, such as time. For example, time was used in each specific model for the file system category, however, it was applied differently across the implementation of specific models. An intermediate step transformation occurred when there were multiple steps to determine an attribute that was common across the category but was determined differently in each model. For example, in the file system category, there was a need to determine the target directory location, but this was done differently for each implementation. A calculation transformation occurred when the attribute was used in a calculation to determine a forensic attribute value.

Table 3-3. Types of Transformation Rules

<table>
<thead>
<tr>
<th>Transformation Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Mapping</td>
<td>Attribute of specific model (SM) is the forensic attribute of the top-level model (TLM).</td>
<td>bytesPerSector</td>
</tr>
<tr>
<td>Equivalent Mapping</td>
<td>SM attribute is equivalent to TLM forensic attribute.</td>
<td>time</td>
</tr>
<tr>
<td>Intermediate step</td>
<td>SM attribute is utilized in an intermediate step to obtain the TLM forensic attribute.</td>
<td>To determine file location need target directory</td>
</tr>
<tr>
<td>Calculation</td>
<td>SM attribute is utilized in a calculations to obtain the TLM forensic attribute.</td>
<td>dataUnitSize = bytesPerSector*Sector sPerDU</td>
</tr>
</tbody>
</table>

*Develop use-case profile (step 7).* Use-case profiles for each use-case were developed from the top-level model of the use-case. The functional grouping implementations were captured as stereotypes in the profile diagrams. The top-level model data types
were used to assist in developing the stereotype tags. However, not all top-level data types were utilized, specifically if they were too detailed or implementation specific.

*Develop/integrate DF profile (step 8).* The use-case profiles were analyzed to create one digital forensic profile. Common stereotypes across all the area profiles were then refactored to ensure that one representation worked across all areas. The area profiles were adjusted so as to not duplicate the stereotypes which resided in the common stereotypes. The common stereotypes and the adjusted unique profiles were combined to create the overall digital forensic profile. Metrics were taken to quantify commonality across the use-cases.

*Constructive Method Contribution.* A contribution of this work was the modeling of the relevant computational mechanisms for evidence creation. How this modeling was performed is identified by the constructive method. The constructive method itself has resulted in artifacts and processes which are also contributions of this work. The artifacts and processes are:

1. Identified a process to construct top-level implementation views of computational mechanisms,

2. Identified modeling patterns to catalog commonalties in computational mechanisms, and

3. Introduced a digital forensic profile along with a process to extend the profile.

3.3 Model specification: use-cases and use-case descriptions-(step 1)
Three model specifications were defined for the file system create/delete use-case, the browser, browse and download use-case and the RAM list process/network.
connection use-case. For each of these use-cases, a use-case description and use-case
diagram are defined.

3.3.1 File system forensics

The file system implementations were based on [55] and were augmented with
materials from URI forensic coursework. Figure 3-7 depicts the file system
create/delete use-case and Figure 3-8 provides the use-case description that included
the high-level scenario and the associated forensic attributes.

The file system allocation/delete use-case scenario begins when the suspect
“saves” a new file. At some later point in time, the user deletes the file by moving it
to “trash”. The file system use-case was investigating evidence that was created
during file allocation and file deletion. Evidence of interest included information on
the file itself, times, and information, all of which could be used to find evidence, and
file slack, which could be areas in which data can be hidden.

When modeling a use-case, only the functions that are available for all of the
implementations should be included in the use-case scenario to ensure the consistency
of the results in the top-level model. For example, since the journaling capability is
not available in FAT file system, it is not addressed in the file allocation/deletion use-
case scenario.
3.3.1.1 File Allocation Table (FAT) file system

The major data structures of the FAT file systems include the boot sector, the FAT, the directory structure, and the clusters. The boot sector contains the information
required for the operating system to determine locations of the relevant data structures. File contents are stored as clusters. The directory is represented as a set of tables whose entries contain information on a specific file or a subordinate directory. The FAT table provides information about the clusters in which the file information is stored.

An example of a file allocation and deallocation in the FAT file system are shown in Figure 3-9. In this scenario, the file “root\dir\file1.dat” which is 6000 bytes is to be allocated to a FAT file system which has a cluster size of 4096 bytes. The FAT scenario begins when the OS reads values from the Boot Sector to determine cluster size and the location of key file structures. The OS then reads the root directory to determine the cluster number of the next directory in the path. Once the target directory is found the metadata for the file entry in the target directory is inserted. This includes long file name (LFN) and short file name (SFN), size, timestamps, and the setting of associated flags. The OS then determines the cluster to be used as the start cluster for the file. This cluster is written, and if there is more to write, the next free cluster is determined from the FAT. This process is repeated until there is no additional file information to write. At this point, the write time is updated.

For file deletion the OS determines the location of the target directory, using the same process as described above for file allocation. Utilizing the target directory entry, the target file is located, and the start cluster of the target file is identified. Utilizing the FAT, the entries in the FAT are marked as “empty”, but the contents remain. In addition, the file names in the target directory entry are modified, but not deleted. Times stamps and relevant flags are updated as required.
3.3.1.2 New Technology File System (NTFS)

The NTFS major data structures are the Master File Table (MFT) and Clusters. The MFT entries are composed of attributes, which are themselves complex data structures. There are MFT entries for directories and files and other data structures which are of importance to the file system.

An example of a file allocation and deallocation in the NTFS file system is shown in Figure 3-10. In this scenario the file “root\dir\file1.dat,” which is 4000 bytes, was to be allocated to an NTFS file system that had a cluster size of 2048 bytes. The file allocation began when the OS accessed the boot sector to determine cluster size and the requisite information to process the MFT. In the data attribute within the associated MFT entry, the allocation of clusters were defined in terms of data runs.
data run is a method in defining how an ordered set of clusters can be logically encoded in bits.

The MFT Entry Bitmap was processed to determine an empty MFT entry. This entry had the relevant attributes created or updated. The Cluster Bitmap was used to determine the set of clusters to which the file contents were to be written. The relevant attributes were updated and then the file content was written.

Next, the target directory in which the file resided was updated. Starting from the root directory MFT entry, the entry for the target directory entry was determined by navigating the directory structure. The relevant attributes for the target directory entry were updated.

File deletion is accomplished by starting with the MFT Entry for the root, processing the relevant attributes to determine the MFT Entry for the target directory. From this, the MFT entry for the target file is determined. The target directory is adjusted to account for the deletion of the file. MFT entry in the Bit Map is processed to indicate that the clusters are available.

After file deletion, the file contents are still in the clusters and the pointers to the underlying attributes still exist.
3.3.1.3 Extended (EXT) file system

The implementation details for allocation and deallocation of files in the EXT file system are shown in Figure 3-11. The major data structure of the EXT files system are the Super Block, Block Group Descriptor table, Inode tables, Block Bitmaps, Inode Bitmaps, and Directory contents.

In this scenario, the file “root\dir\file1.dat,” which was 6000 bytes was to be allocated to a EXT file system that had a block size of 1024 bytes. The scenario began when the operating system accessed the superblock to obtain the block size and structural information. Based on this, the OS was able to process the block that contained the Root Directory Inode Table and to process the directory path to the target Block Group and the associated Inode Table. From the Inode table, the block which contained the target directory contents is determined. The unused space in the directory for the targeted file was also determined. The Inode for the new entry is
identified from the Block Groups Inode Bitmap. Utilizing this information, the file information is entered in the directory structure and the values of the file Inode were initialized. Utilizing the Block Bitmap, the file contents were written to the blocks.

To perform file deletion, the OS system starts from the root directory and process the EXT structures in the same way as described above for file allocation to determine the location of the targeted directory structure. The targeted file elements are removed from the block containing the directory contents and the file Inode was deallocated. In addition, the associated entries for the Block Bitmap are deallocated.

At the end of the deallocation process the contents in the blocks still exist.

---

Figure 3-11. EXT File Allocation/Deletion. Adapted from [55].
3.3.2 Browser forensics

Numerous sources [56] [57] [58] [59] [60] were utilized for the browser model. To gain further insight on how to model the browser user-case additional research was accomplished utilizing the open source database browser for SQLite [61] to analyze how Chrome stores web information. Browser forensics utilizes numerous artifacts, including:

- History: list of visited URLs.
- Cookies: the cookies and associated information from sites visited.
- Downloaded files: the metadata for the files that have been downloaded by the browser.

Figure 3-12 depicts the Browser use-case and Figure 3-13 provides a description of the use-case, which includes the scenario and the associated forensic attributes. This use-case depicts the scenario in which the suspect is either browsing sites (some with cookies enabled) or downloading artifacts. In this scenario, the user (i.e., the suspect) browsed numerous websites, some of which used cookies, and cookies were enabled on the suspect’s computer. The user also downloaded some documents.

Types of evidence include the URL, title of the webpage, time visited, and the number of times visited. Since some of the sites use cookies, the cookie names and cookie values are also available. In addition, the time of cookie creation and access times are available, as is the cookie expiration times.
The underlying information for the browser application was either stored in databases or files in the file systems. Table 3-4. Browser Configurations provides the names of the files and databases that were of interest for the browsers and attributes for this work.
<table>
<thead>
<tr>
<th>Browser</th>
<th>Download</th>
<th>Path</th>
<th>Database</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>Download</td>
<td>\users\user\AppData\Local\Google\Chrome\UserData/Default\databases</td>
<td>History</td>
<td>downloads</td>
</tr>
<tr>
<td></td>
<td>history</td>
<td>/users/user/AppData/Local/Google/Chrome/UserData/Default</td>
<td>History</td>
<td>url</td>
</tr>
<tr>
<td></td>
<td>cookies</td>
<td>/users/user/AppData/Local/Google/Chrome/UserData/Default</td>
<td>Cookies</td>
<td>cookies</td>
</tr>
<tr>
<td>Edge</td>
<td>Cookies</td>
<td>/users/user/AppData/Local/WebCache/WebCacheV01.dat</td>
<td>WebCacheV01.dat</td>
<td>CookieEntryEx</td>
</tr>
<tr>
<td></td>
<td>downloads</td>
<td>/users/user/AppData/Local/WebCache/WebCacheV01.dat</td>
<td>WebCacheV01.dat</td>
<td>Containers</td>
</tr>
<tr>
<td></td>
<td>History</td>
<td>/users/user/AppData/Local/WebCache/WebCacheV01.dat</td>
<td>WebCacheV01.dat</td>
<td>Containers</td>
</tr>
<tr>
<td>Safari</td>
<td>Cookies</td>
<td>/Users/&amp;USER/Library/Cookies/cookies.plist</td>
<td>NSDictionaryCookies</td>
<td>NSDictionaryCookies</td>
</tr>
<tr>
<td></td>
<td>downloads</td>
<td>/Users/&amp;USER/Library/Safari/DownLoad.plist</td>
<td>DownLoad.plist</td>
<td>NSDictionaryDL</td>
</tr>
<tr>
<td></td>
<td>History</td>
<td>/Users/&amp;USER/Library/Safari/History.plist</td>
<td>History.plist</td>
<td>NSDictionairesHistory</td>
</tr>
</tbody>
</table>
3.3.3 Random Access Memory (RAM) forensics

Ligh, Case, and Walters [62] described Volatility tool suite, a well-known open-source tool suite for memory forensics. Volatility supports analysis on the Microsoft Windows, Mac OS X, and Linux operating systems. In addition to the Volatility sources [62], standard OS textbooks and educational sources [63] [64] [65] [66] were utilized for the implementation description.

RAM forensics is the analysis of volatile memory in computer systems. Memory forensics requires a memory image to be captured that provides the state of RAM at a given point in time. In the case of Volatility, an image is loaded into the framework that includes commands to retrieve the desired information. This image is run with a library implemented in Python.

RAM includes the current state of the operating system, executing applications, and network connections. RAM also provides the internal details of data structures. Examples of information memory forensics can identify include [67]:

- Past and current network connections;
- List of running processes at the time of RAM capture;
- User names and passwords;
- Loaded Dynamically Linked Libraries (DLL);
- Open registry keys for a process;
- Open files for a process;
- Unpacked/decrypted versions of a program; and
- Memory resident malware.
Understanding RAM forensics requires an understanding of the operating system data structures and algorithms. From a high level of abstraction, the operating system implementation is described in numerous textbooks which were previously cited. Typical operating system textbooks segment the operating system into five categories: Process Management, Memory Management, Virtual Memory, Storage Management, and I/O. However, specific information on any implementation, such as Windows and iOS, are scarce, and the differences between implementation versions are even less available. To that end, forensic information has only been identified by researchers who have been able to reverse engineer the operating systems. Open source operating systems, such as Linux, present a different type of issue. With open source the source code is likely available, however there is significant work required to understand the source. The important aspects of the operating system need to be abstracted so the forensic analyst knows what to look for.

Figure 3-14 depicts the RAM use-case diagram and Figure 3-15 provides the associated use-case description that identifies the high-level scenario and the forensic attributes. This use-case was used to track process and network connections associated with applications. In this use-case there were multiple processes running in the computer and open network connections. The user (i.e., suspect) hacked into a target machine. The inspector was trying to determine if malware was installed and/or if information was being compromised or exfiltrated. Types of information of interest were the name, process identification, process parents, and times when the processes were started.
Figure 3-14. RAM Use-case Diagram

Figure 3-15. RAM Use-case Description

RAM (Process/Connection) Use-Case

Targeted Mechanisms: Process & Connection Information

Scenario: The investigator uses a captured segment of memory to determine the following:
- What processes were running
- Is there any obvious indications of malware
- What sockets and ports were open and what are active

- Forensic Attributes: processes/timelines:
  - The list of running processes (name and PID) will identify what applications were running on the system.
  - The time identifies with the process started. Based on these forensic attributes time lines may be generated.

- Forensic Attributes: signs of malware
  - The list of running processes may provide insight if there was malware on the system. This can be seen with regard to the process names along with the child process spawned by the parent process. Malware can be spawned by infected processes.

- Forensic Attributes: Open Ports (local port, remote port)
  - Open ports provide opportunities for intruders to gain access to the system.

- Forensic Attributes: IP Activity (local IP, remote port)
  - Provides evidence on what other cites the system has been communication to.
For this use-case, the focus was on running processes and network connections. Utilizing RAM forensics enabled the stakeholder to understand the processes that were being executed on the device. Knowing the processes that were running provided a forensic analyst information about what applications, and potentially what activities, were being performed. A forensic analyst could potentially determine if there were any harmful processes running since the malware processes were likely to be hidden or disguised.

Knowing about any open connections provided evidence of whether the device was communicating to the outside or if an outside source had hacked into the device. IP addresses provided information on the location of the outside source. Knowing about any open ports provided information on potential vulnerabilities.

Figure 3-16 is a compilation of a generalized high-level operating system architecture derived from numerous sources [63] [64] [65] [66]. The operating system architecture in the figure is a very high level abstraction which was used for the development of the RAM model. Figure 3-15 depicts the relevant operating system components for the executing application process (e.g., Process). Processes have virtual memory space in which their stacks and heap reside. Application executables are loaded into the process memory space. The application has a network connection. The operating system has a Process Management and I/O Management subsystems. The Process Management subsystem provides the algorithms that service application processes. The application has a Handle Table that provides the reference to the network communication stack (e.g., CommsObj) between the application and the
Network Interface Card (NIC). The application communicates over the network utilizing the Socket API.

The Process List has a reference to all the processes. The Network Connection List has a reference to all the connections, with each connect having a CommsObj. There are two types of command line tools. The Process Command Line Tool provides the list of processes and relevant forensic information for the processes. The Network Command Line Tool provides a list of connection and the relevant forensic information associated with the connection.

Figure 3-16. RAM Architectural View

Processes are described by a structure that contains relevant details and references to other key operating system structures/resources. The process list, which contains the process structures, can be implemented as a linked list. For example, an implementation for a Windows 7 system is shown in Figure 3-17. However, based on
general textbook descriptions it is reasonable to conclude that a link list of processes could also be utilized to represent Linux and OS.

![Diagram of process list]

Figure 3-17. Process List. Adapted from [62].

By analyzing the process list, the forensic analyst is able to determine the applications that are running and potentially the start times for the applications both of which could be tied to user activity. In addition, the forensic analyst may want to inspect for malware. This can be accomplished by:

- inspecting for unexpected processes, or
- inspecting the parent and child to determine anomalies in how the child was launched.

The application utilizes the sockets API to interface with the Communication Object, which represents the network communications of the system. A socket provides an “endpoint” for sending and/or receiving data. A socket can be accessed by a process to either receive or send data. The process knows how to address the socket by accessing the file descriptor in the handle table, which is the implementation of the OSI-7 layer stack. The network stack contains the network information, such as IP addresses and ports for the application.
3.4 Model implementation (steps 2 & 3)

3.4.1 Modeling overview

The UML is a modeling language based on an Object Oriented (OO) programming approach for software development. UML modeling is equivalent to programming in most procedural languages, but modeling is at a higher abstraction level. UML models are often used to provide the overarching software design for software systems. This design is then utilized to implement the software system in the more expressive procedural languages. In addition, models, like traditional source code, can be executed utilizing a model compiler.

UML was derived from previous methodologies in the 1990s under the OMG standards organization. UML is a visual language based on diagrams. The objective of UML was to address complexities in software design and architecture. UML provides a formalism to model structure and behavior to allow for the generation of various consistent views from a single model representation. Over time, UML has been extended to address vocabulary and concepts in other domains through profiles.

UML is in a family of modeling standards. Standards are significant in that they provide the requirements for how the technologies are to be implemented and ensure that the implementations are consistent. These standards provide a means for tool and environment vendors to develop products to perform modeling. Standards also ensure that the modeling constructs are the same across environments, which allows a consistent utilization of models. The standards are also implemented by tool vendors or organizations developing open source tools. For example, this work was implemented utilizing publically available implementations of tool environments. The two primary environments to support the modeling for this work were eXecutable
Translatable UML [21] and Papyrus [68]. The model execution utilized xtUML and the profile development utilized Papyrus.

Papyrus is an open source plug-in to the Eclipse Integrated Development Environment (IDE). Papyrus fully supports the UML specification and as a result allows for the modeling of profiles.

The open source environment utilized to develop the executable models was xtUML, developed by BridgePoint. This modeling environment supports a limited subset of UML and is a plug-in to the Eclipse IDE. The xtUML environment utilizes class diagrams, component diagrams, state diagrams, and an action language and features a model compiler allowing for model execution. In addition, the environment allows the ability to bridge with applications which are outside of the xtUML environment, which allows for model instrumentation in which the model behavior can be captured.

3.4.2 UML diagrams

UML models are created (i.e., programmed) as the UML diagrams are developed. In programming languages, grammars are used to define the language, whereas in modeling languages, it is the underlying metamodel which defines the modeling language. The modeling environment enforces the metamodel rules which are required to ensure diagram correctness and consistency.

UML consists of fifteen diagrams, seven structural and eight behavior (depicted in Figure 3-18). Structural diagrams such as the class diagram represent the entities being modeled and the relationship between the entities in terms of
associations. Another structured diagram is the object diagram, which shows specific instances of the classes. Instances are related by links.

An example of a behavioral diagram is the state machine diagram, which shows the internal behavior of the class. The use-case diagram, another behavioral diagram, identifies system functions which are contained in the system boundary along with the actors or stakeholders who utilize the system based on the use-case functions.

To be valid, a diagram needs to adhere to the UML-specified rules. These rules ensure that the model representation is consistent across the diagrams.

![Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml)

This effort was specifically focused on the class diagrams, state machine diagrams, use-case diagrams, and, to a lesser extent, object diagrams. In this work, object diagrams were not developed but their equivalence was realized and viewable in the modeling environment when the models were executed.
**Class diagram: structural formalism.** A class diagram consists of classes and relationships between classes. A class is a template used to describe objects that consist of properties or attributes and behaviors defined by methods or operations. Classes, when instantiated in a program, become objects.

Relationship between classes are shown as associations, which are represented as lines between classes. Association end points may have role names and multiplicities. If a role name is not specified, the role name is defaulted to the class name, in lower case, at the end of the association. The arrow on the association lines represents how the class diagram is navigated. Navigation shows the sequencing of how class attributes are to be accessed. Multiplicities are also shown on either end of an association. Multiplicities indicate the number of instances that result when the class is instantiated. Multiplicity values are often represented as 0, 1, and ‘*’. A multiplicity of 0 to any number of elements is denoted by ‘*’ and a multiplicity of 1 to any number of elements is denoted by 1 ‘*’. Cases in which instantiation results in multiple objects of the target class result in a collection of objects.

A collection may be a set (e.g., a mathematical set with no duplicate elements), a bag (e.g., a set which may have duplicate elements), an ordered set (e.g., elements ordered by position), or a sequence (e.g., a bag in which elements are ordered).

Important association types include:

- Inheritance (represented as an arrow): provides a subtyping (‘is-a’ relationship);
• Composition (represented as a filled in diamond): an association type between class elements which have a “has-a” or “part-of” relationship;

• Aggregation (represented as a diamond): an association type between class elements which have a “has-a” or “part-of” relationship. This differs from composition in that the “parts” can exist even if the whole does not;

• Dependency Association (represented as a dashed line on an association to a dependency class): a class which provides attributes on an association.

Figure 3-19 provides an example of a class diagram notation. Class diagrams may also contain data types, as can be seen in the diagram. Data types are similar to classes but do not have operations. In this example, class A and class B have attributes and operations; attributes have types and multiplicities. Attributes which have multiplicities greater than one can be viewed as a collection. Attributes which are of a type of a specific class or data type have a dependency on that class or data type. As an example, the class B attribute attC can have an array of zero or more of KeyValue data types. Class B inherits from class A resulting in class B having attributes of attS and attC. There is a composition between with class B and class C. There is a dependency class, class F, on the association, between class B has and class D. Class E has a shared aggregation with class G. In addition, both role names and multiplicities are shown on the associations.
Another example of a class diagram in Papyrus is shown in Figure 3-20. The DataElementCollection class has a size attribute. In addition the DataElementCollection class is composed (associated) with the DataElement class with a 1-to-many relationship. The DataElement class has an id attribute of type Integer and a keyValue attribute of type KeyValue, which is a data type. The attribute KeyValue data type has a tuple of key of type Integer and a value attribute of type Integer.

The diagram describes a collection class of unordered DataElement, with each DataElement containing a key-value pair. The DataElementCollection class has the following functions:

- addE: to add a key-value pair,
- deleteE: to delete a key-value pair.

The DataElement has the following functions:

- read: to provide the value for a specific key,
- write: to modify the value which is associated with an existing key.
Figure 3-20. Example Collection Class

*Use-case diagrams.* A use-case diagram consists of a system of interest, actors, use-cases and, associations. The system defines the boundaries of the entity being modeled; in the case of this work, the system is the forensic device of interest. Actors represent user roles or types of external systems. The system to be modeled consists of use-cases. A use-case is a set of actions (e.g., activity) which performs a system function. The actors interact with the system through associations. Two association types specific to use-cases are include and extend. The include relationship shows a dependency of one use-case on another use-case and the extend relationship specifies additional use-case which only occurs during certain scenarios.

In the example shown in Figure 3-21, the actor is the suspect. The system itself is a device type of Computer. The actor chooses one of the use-cases, Browse Internet, Save file, or Start application. All these use-cases depended on operating system use-cases and therefore the extend relationship is used. The Operating System, the Word Application and the Browser use-cases are grouped as components in the
Computer. **The Browse Internet use-case is, extended by two use-cases, Enable cookies and Download file.** These use-cases occur when the user enables cookies, if required by the website or if the user downloads a file. In addition, the user may browse to a website and decide to download a file. These use-cases are contained in the Browser component. Also depicted in a diagram is a Word Application which has a use-case to Start application or Save file.

![Diagram](https://via.placeholder.com/150)

**Figure 3-21. Example Use-Case**

**State machine: dynamic formalism.** A state machine is a behavioral model that consists of a set of states. A state is some condition of the system at a particular time. The current state can transition to another state given an event. There are typically two special states, the initial state and the end state.
Figure 3-22 provides an example of a state machine depicting the state in which a computer could be from a user perspective. Initially the user logs in. Once logged into the system, the condition of the system will allow the user to start interacting with the system. The user may choose to start a browser application to browse the Internet, or to start an application to work on a document, or they might choose to install an application. The state machine provides transitions to go back to the LoggedIn state through which the user may choose another activity or simply log out. The state machine defines the set of states that may be of interest to a digital forensic stakeholder. Each of these states may have associated attributes that might contain evidence.

Figure 3-22. Example State Machine
3.4.3 Profile extensions for domain representation

A profile is a mechanism to extend UML to add constructs that address domain vocabulary and concepts. This is accomplished by extending the UML metamodel with stereotypes. Stereotypes themselves may contain tags that are properties of the stereotypes. OCL is utilized to further specify the relationships between the domain specific elements by defining constraints.

The OMG modeling standards define various levels of modeling. To understand how profiles relate, it is helpful to understand how they relate in the hierarchy. The OMG model-driven engineering is a development methodology to create conceptual models in a domain. The OMG Meta Object Facility (MOF) is a standard for model driven-engineering. The MOF Metadata Architecture is shown in Table 3-5. This layered architecture consists of four levels. The highest level, M3, is the most abstract and provides the metamodel for UML. UML is at level M2 and provides the metamodel rules for the models which are developed in a UML environment. This is the level of modeling performed in this work. The instantiation of these models resulted in the actual objects that related to some real world entity. When the model was executed, instantiations were created consistent with the scenario; this is at the M0 level.

<table>
<thead>
<tr>
<th>Meta-level</th>
<th>MOF Terms</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>Meta-metamodel</td>
<td>MOF Model</td>
</tr>
<tr>
<td>M2</td>
<td>Metamodel</td>
<td>UML Metamodel</td>
</tr>
<tr>
<td>M1</td>
<td>Model</td>
<td>UML Model</td>
</tr>
<tr>
<td>M0</td>
<td>Object</td>
<td>Modeled Systems</td>
</tr>
</tbody>
</table>
This work extended the UML metamodel by identifying a profile that addressed the digital forensic domain. The models developed in this work were at the M1 level. In a UML modeling the metamodel, instead of a grammar in traditional languages, is used to define model semantics.

A domain specific profile was created by extending already predefined UML elements. More specifically, a profile stereotype extends the meta-class element of the metamodel. The meta-class could be any of the elements in a UML model (e.g., class, association, state machine, etc.).

The elements in the metamodel are associated with each other by a class diagram. A profile does not alter any of the elements of the metamodel (i.e., no associations are changed) but extends these associations with stereotypes. The stereotypes themselves define new associations in the profile. These associations may need additional constraints, which are added using OCL. In addition, the stereotypes may contain meta-level properties that are identified with tags.

OCL is a declarative language that can add constraints to UML diagrams, such as class and state diagrams. OCL is utilized to further specify the meanings in the diagrams to which it is applied. OCL can be viewed in terms of set theory. OCL has a standard library of primitive types and provides operations for both primitive and user-defined collection types. An OCL expression has a context that is the element for which the OCL expression is defined. To get to another element, the expression can have navigation rules to reach another element.
An example of a profile is displayed in Figure 3-23. The stereotype names relate to terminology describing components (e.g. OS, File System) which are of interest to digital forensic stakeholders. In this figure the OS stereotype extends from both the UML Class and UML StateMachine metamodel element. The FileSystem stereotype extends the UML Class metamodel-element and Oscall stereotype extends the UML Association metamodel-element. The OS stereotype has tags of name and version which are of type string and has an association with the FileSystem stereotype. The FileSystem stereotype has a clusterSize and name tags, both of type String. There are two OCL constraints, both having a context of FileSystem. The SizePositive constraint requires the cluster size to be greater than zero and the NamesLetters constraint requires the name of the FileSystem to consist of letters only.

Figure 3-23. Files System Area Profile
### 3.4.4 Executable models

**Bridgepoint xtUML environment.** An executable model is a model that has run-time behavior. This requires that the behavior of the model be sufficiently specified. Executable models utilize behavior diagrams such as state or activity diagrams or an action language for control flow. In order to adequately specify executable models, additional operational semantics were added to some UML diagrams using the fUML standard. The textual representation assists in the development of executable models in cases in which a total graphical representation could be difficult. The OMG has identified action languages semantics to UML standard. The action language is model aware in that it can interact with model elements to enable the execution of the behavior diagrams. The action language for xtUML is the Object Action Language (OAL).

The modeling environment provided by xtUML is based on Shlaer-Mellor modeling methodology, which uses a subset of UML notation. This modeling methodology only utilized a subset of UML diagrams and there are some minor differences within the constructs of these diagrams that needed to be addressed for this work. As an example, xtUML state machines do not have specific initial and final state symbols.

The key elements in the executable model for this work, the class diagrams, the state machines diagram, and the OAL action language, are depicted in Figure 3-24. The class diagram is utilized to model the forensic data structures and key components of interest. The associations between classes are used for navigation by the OAL. The
A state machine diagram is utilized to model the behavior of the system for the scenario of interest. The OAL provides the constructs to enable the control flow.

Figure 3-24. Model Element Types

An example of a class diagram in xtUML is shown in Figure 3-25. This is the same diagram shown in the papyrus example (Figure 3-20), with some important differences. As an example, the xtUML class diagram does not distinguish between the different types of associations. For this work, the differences between xtUML and Papyrus did not affect the outcome because they were mostly notational.
In the example, the OS class has an attribute, `osContext`, which is a nested data type of `OSData` is the type of attribute `osContext`. Figure 3-26 shows the structure of the nested data type as modeled.

Figure 3-25. xtUML Class Example
OAL is model aware, that is, it is able to interact with the model elements. The action language provided control flow between classes and allowed access to the class attributes. As an example of OAL, the operations in the class diagram snippet are shown in Figure 3-27.
Figure 3-27. OAL Operation Implementation Examples

```c
DataElementCollection::addE(key:string, value:string): void

create object instance de of DataElement;
de.kv.key = param.key;
de.kv.value = param.value;
select any deCollect from instances of DataElementCollection;
relate deCollect to de across R2;
```

```c
DataElementCollection::deleteE(key:string): void

select any de related by self-> DataElement[R2]
    where selected.kv.key == param.key;
select any deCollect from instances of DataElementCollection;
unrelate de from deCollect across R2;
delete object instance de;
```

```c
DataElement::write(value:string): void

self.kv.value = param.value;
```

```c
DataElement::read(): string

return self.kv.value;
```
In Figure 3-25, the OS class has an associated state machine symbol. The state machine (behavior) diagram consists of a set of states and the state transitions (see Figure 3-28). Each state contains OAL to modify attributes, to make function calls, and to transition to other states. In the case of the example, the \texttt{init} transition triggers the state machine. In state $X$ there is OAL code to invoke the functions associated with the \texttt{DataElementCollection} class after which there is a transition to state $Y$. The state machine along with the OAL provide control flow for the model.

![State Machine Example](image)

**Model execution.** In this work control flow of the executable model was defined in the OS or application state machines for a particular scenario. The OS and applications interacted with the various modeled forensic data structures during execution.

The model needed to be initialized before it executed. This initialization is analogous to the instantiation of classes in an object oriented languages or the equivalent of the object diagram in UML.

In the context of this work, the model configuration was based on the use-case scenario being modeled. The model contained a function that initialized the scenario by instantiating classes and defining links between the instances. To configure the
scenario, an initialization function needed to be developed. Figure 3-29 shows a configuration function for the example. This function was manually invoked.

After the configuration function was executed, the instantiated classes and relationships can be seen in the xtUML display environment, shown in Figure 3-30. The DataElement instances and the shading of the Init state along with the instance values were of note.

Next, after being initialized, the start function was invoked. This function generated an event which triggered the state machine as illustrated with this OAL snippet:

```
create event instance evt of OS_A1: init() to OS class;
generate evt;
```

This trigger event caused the state machine to go through its states and to execute the OAL in the states to which it transitioned. Figure 3-31 shows the state machine in its final state, Complete.
create object instance os of OS;
create object instance deCollect of DataElementCollection;

create object instance de1 of DataElement;
create object instance de2 of DataElement;
create object instance de3 of DataElement;

de1.keyValue.key=1;
de1.keyValue.value=101;

de2.keyValue.key=2;
de2.keyValue.value=202;

de3.keyValue.key=3;
de3.keyValue.value=303;

relate os to deCollect across R1;
relate deCollect to de1 across R2;
relate deCollect to de2 across R2;
relate deCollect to de3 across R2;

Figure 3-29. Example xtUML Scenario Creation

Figure 3-30. Example Initial State
Capturing model behavior. The models were instrumented to record model behavior. Changes in modeled forensic data structures, state transitions, and changes in the focus of the thread of control are examples of captured behavior. A Java application outside of the model recorded these changes in an XML file (see Figure 3-32).
Figure 3-32. Capturing Behavior
More specifically, the xtUML environment provided a bridge or external entity object from which model operations could pass information to an external program. Figure 3-33 provides a snippet showing the definition of an external entity. This external entity defines two operations, linkData and attributeDataI. The operation linkData provided information on the entity with the control flow (e.g., the operating system) to the entity on which it was focused (e.g., forensic data structure). Also provided in this operation was the “message” that might be passed between the entities. The attributeDataII operation was used to pass the state of the model data structure attribute to be recorded in an XML file when it was read or written during the scenario. The defined operations were invoked from the model (see Figure 3-34). The operation implementations were static methods in the associated Java application (see Figure 3-35). A snippet of the resulting XML file is shown in Figure 3-36.

![Example xtUML Bridge](image)

**Figure 3-33. Example xtUML Bridge**

```xml
<operation>
  linkData(toInstance : string, fromInstance : string, message : string) : void
  attributeDataI(instanceName : string, attributeName : string, value : integer) : void
</operation>
```

**Figure 3-34. Example xtUML External Call**

```java
XMLBridge::linkData(toInstance:clusterDataUnitCol,_instanceName,fromInstance:oName,message:"write");
XMLBridge::attributeDataI(instanceName:oName,attributeName:"localFS.leftToSave",value:oContext.localFS.leftToSave);
```
public static void attributeData(String instanceName, String attributeName, int value) {
    try {
        PrintWriter writer = null;
        FileWriter fw = new FileWriter(path + fileName, true);
        writer = new PrintWriter(fw);
        writer.println();
        writer.println("<step>");
        writer.println("<toInstance>");
        writer.println("<empty>");
        writer.println("<toInstances>");
        writer.println("<fromInstance>");
        writer.println("<empty>");
        writer.println("<fromInstance>");
        writer.println("<message>");
        writer.println("<empty>");
        writer.println("<message>");
        writer.println("<instanceName>");
        writer.println("<instanceName>");
        writer.println("<attributeName>");
        writer.println("<attributeName>");
        writer.println("<attributeValue>");
        writer.println("<attributeValue>");
        writer.println("<value>");
        writer.println("<attributeValue>");
        writer.println("</step>");
        writer.println();
        writer.close();
    } catch (Exception e) {
    }
}

Figure 3-35. Snippet to Create XML File

```xml
<step>
  <toInstance>BootSector</toInstance>
  <fromInstance>OS</fromInstance>
  <message>retrieveBootStruct</message>
  <instanceName>OS</instanceName>
  <attributeName>sectorsPerFAT</attributeName>
  <attributeValue>953</attributeValue>
</step>
```

Figure 3-36. Sample XML Snippet
3.5 Metrics utilized

This work proposes two metrics, the Top-Level Model Abstraction Metric (TLAM) and the Implementation Commonality Metric (ICM) metric. The TLAM metric assesses top-level model abstraction. The ICM metric identifies the degree of commonality within implementations, across implementations in use-cases, and between use-cases.

This work addressed complexity in a manner similar to that which software engineering addresses complexity, specifically by utilizing abstraction. The premise was that a top-level model with fewer elements is less complex and more readily understandable than a specific implementation model with more elements. A common implementation pattern is likely to be an abstract view which represents multiple unique implementation elements. To reduce model complexity, it is advantageous, to maximize the number of common, abstracted implementation patterns, and minimize the number of unique elements. There are two cases of interest to illustrate this point, comparing the top-level implementation with a specific implementation and comparing the top-level implementations against all the individual implementations in a use-case.

Figure 3-37 illustrates the TLAM metric and the two cases of interest. The top of the figure shows a top-level model with only two elements. The bottom of the figure shows three functionally equivalent implementations, each composed of unique elements. The functionality of the implementations is the same as the functionality of the top-level elements. However, the implementations are different from each other. In addition, there are less elements top-level elements suggesting that the top-level
elements are at a higher abstraction level. The TLAM for case 1 is a percentage of top-level model elements to individual model elements. TLAM is calculated by:

\[
\text{TLAM}\% = (1 - \frac{\text{TLME}}{\text{IME}}) \times 100 \quad (\text{equation 1})
\]

where:

- TLME is the total number of top-level model elements
- IME is the total number of implementation model elements

This perspective identifies the abstraction level of the top-level for a specific implementation. As an example, if the top-level model represented file system functions, it could be used to compare the level of abstraction to the NTFS implementation. In the example in Figure 3-37, it is seen that the elements in the top-level model result in a 67% reduction of the elements when compared to implementation 1 (I1), 33% reduction when compared to I2, and a 60% reduction of elements when compared to I3. The concept is that it is advantageous, less complex, and easier to understand the fewer abstracted elements of the top-level model versus the numerous elements in an implementation specific model.

TLAM case 2 compares the top-level implementations against all the individual implementations in a use-case. This is a ratio of the individual model elements to the top-level model elements. This perspective identifies the abstraction level to all implementations across a particular use-case. For example, in the file system example, this may be comparing the number of elements in the FAT, NTFS, and EXT implementations for specific functionality. Looking across implementations
there may be additional increases of abstraction realized in TLAM case 2 since the
top-level model provides an abstraction for multiple implementations vice one
implementation. In the case of the example in the figure, there is a 7:1 reduction. It
should be noted that the smallest ratio, will be N:1, were n is the number of
implementations. This will be the case when all the implementations and top-level
model contain the same number of elements. This will occur when all the
implementations and top-level model are the same.

Figure 3-37. TLAM Illustrated

The ICM metric is analogous to the reuse leverage metric of [69] and the
abstraction metric [70]. The elements that these metrics were measuring are identified
in Table 3-6. This work argues that the ICM is analogous to the reuse metric and
abstraction metric and as a result the ICM metric utilizes a similar calculation. The
values for all the metrics in the table range from 0 to 1 and are non-dimensional. In
the case of the ICM 0 designates no commonality across implementations and 1
designates 100% commonality across implementations.

Table 3-6. Related Metric Types

<table>
<thead>
<tr>
<th>Metric</th>
<th>Element Type of Interest</th>
<th>Other Element Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuse Leverage</td>
<td>Reused Objects</td>
<td>Built Objects which are not reused</td>
</tr>
<tr>
<td>Abstraction</td>
<td>Abstracted Classes</td>
<td>Concrete Classes</td>
</tr>
<tr>
<td>Implementation Commonality</td>
<td>Common Patterns</td>
<td>Unique Elements</td>
</tr>
</tbody>
</table>

The Reuse Leverage metric measures the reuse of components across a
software project. The Abstraction metric measures the degree of abstraction across a
package. The Reusability Leverage metric determines the reuse of “objects” in a
software system. The objective is that if there are fewer “objects” to reuse, there are
advantages in the costs of developing and maintaining less “objects”. The abstraction
metric identified the degree to which concepts were abstracted from the details. The
ICM is analogous in that the patterns abstract the details of the unique
implementations. The reuse and abstraction metrics are of the form:

Metric = EoI/TE

where:

- EoI are the element types of interest
- TE are the total elements in the population

Using this form, the proposed ICM for measuring commonality in a single
implementation or between two implementations can be expressed as:
ICM = CIS/TIS  \textbf{(equation 2)}

where:

- CIS is the total number of common implementation structures (e.g. common patterns).
- TIS is the total number of implementation structures.

An implementation structure is the modeled data structure and may consist of one or more model elements. The specifics of how CIS and TIS are determined is dependent on how the ICM is utilized. In each case, it is key to determine the common set of structures for the entities which commonality is to be determined. Figure 3-38 identifies the three cases in which the ICM metric is utilized. These cases are:

- Case 1: Common patterns utilized in a specific implementation.
- Case 2: Common patterns utilized across a use case.
- Case 3: Common patterns between use-cases.

Figure 3-38. ICM Illustrated
The application of the ICM equation was as follows:

**Case 1 (single implementation).** In a single implementation equation 2 was utilized where:

- CIS: the total number of common implementation structures in the implementation. The CIS patterns need to be previously defined for this calculation. As an example the common patterns may be those which are defined for the use-case.
- TIS: the total number of implementation structures in the implementation.

The common patterns in which the implementation measured against will need to be defined. In the case of the example in the figure, the implementation consists of four model elements, two of which are identified as common. The resulting metric is 1/2

**Case 2 (across a use-case).** Across a use-case of multiple implementations, the metric is calculated, utilizing equation 2, by comparing two implementations at a time. The metric is the average of applying equation 2, \( _nC_2 \) times, where \( n \) is the number of implementations. More specifically:

\[
ICM = \frac{1}{n} \sum_{i=0}^{m} \frac{\text{CIS}_i}{\text{TIS}_i} \quad \text{(equation 3)}
\]

where \( m = _nC_2 \)

In this context:

- CIS: the common implementation structures across the two implementations.
• TIS: the total implementation structures between the two implementations.

This metric can also be applied to additional levels of functionality within the implementations. As an example, in this work, the metric was applied to functional groups to provide additional granularity.

In the example in the figure, there are three implementations being compared. When I1 and I2 are compared, C1 is the only common structure between the implementations. This common structure occurs twice. In addition, there are a total of ten structures resulting in a metric calculation of 1/5 comparing the two implementations. This is repeated for the other two combinations. The average of these calculations result in a value of 2/5 which defines the ICM for the use-case

**Case 3 (comparing two use-cases).** Across use-cases equation 2 as utilized where:

• CIS: the common implementation structures across the use-cases.

• TIS: the total implementation structures in both use-cases.

In the example of the figure, the use-cases have three common patterns, C1, C2, and C3. These common patterns occur fifteen times. There are a total of thirty elements which resulted in a metric calculation of ½.

**Metric Contributions.** A contribution of this work was the identification of metrics to support commonality and abstraction analysis. This was accomplished with the establishment of the TLAM and ICM metrics.
3.6 Animation Application Implementation

A prototype animation application depicting the file creation/deletion scenario for the FAT file system was created. The application showed the key attributes of evidence creation for the relevant underlying computational architecture. The application utilized a script generated by executing a model for the given scenario. The application provided views to meet the needs of different stakeholder roles.

The animation application read in the behavioral XML script from the FAT scenario. The application animated the behavioral script. The animation showed the control flow of the operating system by showing the forensic data structures that were being read from or written to as the operating system changed states through the scenario steps. The control flow was depicted by an arrow that indicated the data structure that was being read or modified.

The architecture for the animation application is shown in Figure 3-39. The application utilized three XML files that were generated from the model. The initial.xml file provided the initial settings from the model configuration files. The detail.xml file populated data structures to provide additional information as required, and the script.xml file provided the detailed steps.

The animation application consisted of the following functionality: parser, data structures, animation logic, and the Graphical User Interface (GUI). The parser extracted the information from the XML files and populated the application data structures. Based on the user-controlled GUI settings and the parsed script, the animation logic provided the mechanisms to display the information based on the user selections and actions to step through the display. The GUI provided the requisite displays of the animation. The user was able to select various levels of detail to view
via the GUI. For example, detail information was provide to identify the byte location and additional descriptive information on the forensic data structures.

Figure 3-39. Visualization Architecture

A demonstration of the prototype animation application was provided to the URI DFCSC. This demonstration provided a venue for expert review to determine how closely the application represented the forensic concepts being modeled and to allow feedback on the potential applicability of modeling to the field of forensics and stakeholder roles.

**Model to Application Contribution.** A contribution of this work was to generate computable model artifacts to facilitate the execution of a relevant digital forensic animation application.
CHAPTER 4

4 FINDINGS

This chapter details the findings of this work. The findings included:

- Pattern identification (step 4).
- Functional grouping definitions (step 4).
- Top-level general models and forensic area profiles (steps 5 – 7).
- Top-level forensic profile (step 8).
- Constructive procedure for extensibility.
- Metric results.
- Application analysis.

This chapter discusses the model implementation patterns and functional groups resulting from the analysis of the specific models. The functional groupings provided a finer grain decomposition of the functions defined in the scenario use-cases. The constructive analysis resulted in the definition of the top-level models along with their associated profiles. Additional constructive analysis of the use-case profiles yielded the top-level profile. This overall, repeatable, approach was defined in the constructive procedure. Metrics were recorded to assess the resulting commonality across models. Lastly, the results of the animation application will be discussed.

4.1 Specific model Analysis (step 4)

4.1.1 Pattern analysis

The two types of patterns of interest were structural static patterns and behavioral patterns. The static patterns described the underlying architectural data structures of the forensic scenario. The behavioral patterns described the scenario and
the resulting control flow as dictated by the operating system or applications, per the scenario. The patterns were derived from analysis of the implementation models.

4.1.1.1 Forensic static patterns (data structures).

There were six types of (non-unique) static structures which were identified from the analysis of the specific models. Structure was dictated by the classes and the associations between classes. The static structures included:

- Single Elements.
- Unordered Lists (Multiple Elements).
- Ordered Lists (Multiple Elements).
- Multiple List (e.g. Directory/File System).
- Pointers to Lists.
- Amplification of additional data.

These static structures were further refined by considering the behavioral operations that could be performed on these structures and are shown in Table 4-1 as the forensic static patterns. The table identifies the pattern name, the specific structure utilized for the pattern, whether or not that the elements in the pattern needed to be ordered, and the behavior aspects of the patterns. The behavior of the pattern was defined by the operations that could be performed on the classes of the structure, such as being able to add or delete elements or to read/write elements. It also should be noted that each of these patterns could be implemented with or without nested data types. The nested data types provided additional flexibility when modeling complicated data structures.
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Base Structure</th>
<th>Ordered</th>
<th>Size Varies</th>
<th>Modifiable</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static unordered elements non-modifiable (SUEN)</td>
<td>Single Element</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>FAT Boot Sector</td>
</tr>
<tr>
<td>Static unordered elements modifiable (SUEM)</td>
<td>Single Element</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>OS and Applications</td>
</tr>
<tr>
<td>Static unordered list modifiable (SULM)</td>
<td>Ordered List</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>Collection of Clusters/Blocks</td>
</tr>
<tr>
<td>Static ordered list modifiable (SOLM)</td>
<td>Ordered List</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>Bitmaps</td>
</tr>
<tr>
<td>Dynamic unordered list modifiable (DULM)</td>
<td>Unordered List</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>Process and Process File Descriptors</td>
</tr>
<tr>
<td>Dynamic ordered list modifiable (DOLM)</td>
<td>Ordered List</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Process List</td>
</tr>
<tr>
<td>DynamicOrderedTableModifiableOfDynamicUnorderedTablesModifiable (DOTMDUTM)</td>
<td>Multiple List (Directory)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Directory</td>
</tr>
<tr>
<td>Multiple Types Reference (MTR)</td>
<td>Pointers-to-lists</td>
<td>no</td>
<td>N/A</td>
<td>N/A</td>
<td>NTFS:MFTEntry, EXT:Blockgroup</td>
</tr>
<tr>
<td>Detail</td>
<td>Detail</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
</tr>
</tbody>
</table>
In describing the forensic patterns, it is useful to describe each of the structures on which the forensic patterns are based and to identify how these structures are refined by the addition of operations and nested data types.

**Single elements structure.** The single elements contain one or more key-value pairs, as shown in Figure 4-1. The key of the key-value pair has a unique string that is associated with a value that is of type string. The single element structure has the characteristics that it typically does not have multiple instantiations and is static, that is, the number of key-value pairs cannot be added or deleted. Figure 4-2 shows the Single Element pattern with nested key-value pairs. In general, any of the structural patterns may consist of a nested collection of key-value pair collections. In the case of the diagram, the SingleElement has an attribute kvGroup that had a type of KeyValueGroup that itself has one or more attributes of type KeyValue. The notion of nested types allows modeling flexibility of more complicated data structures. Note that OCL constraints are shown to ensure that there are no duplicates in the key-value pairs and key-value groups.
As an example, this pattern applied to the FAT boot data structure is shown in Figure 4-3. In the case of the FAT boot sector, the order of values is not of concern.
This example was the SUEN pattern which was mapped to the table from [55]. The number of attributes did not change (i.e., it remained static). In addition, these values were startup values and were not modifiable.

Figure 4-3. SUEN Example

**Lists structure.** Lists are modeled as collections, as depicted in Figure 4-4. Lists modeled utilizing nested datatypes are depicted in Figure 4-5. The list collection class is composed of one or more Node elements. The Node element consisted of a key-value pair. The key is a unique string that is associated with a value that is of type string. There are three characteristics of a list that are of interest, ordered versus
unordered, constant size versus variable size, and whether the values of the key-value pair are modifiable.

The unordered list is a collection of 1-to-many of the associated type Node. There was no ordering of elements imposed by this structure. Data structures in which order does not matter would utilize this model representation.

In other cases, the list needs to be an ordered. The ordered list contained an association of the collection class to one initial entry element. From this initial element, the next Node can be reached via the next association. This is analogous to a linked list. There is an inherent order in this representation enforced by the structure of the model. As an example, the bitmap data structure was modeled as an ordered list to allow the next element in the sequence to be selected. Specifically, in the EXT block bitmap, when allocating blocks to a file it was optimal to choose the blocks sequentially to prevent fragmentation.

If the collection membership needs to change, the operations to either add or delete Nodes are required. An example of a list that never change size is the FAT table. An example of a list that requires new members to be added or deleted from the collection is the list of operating system processes in a process list.

Also of note was that this pattern consisted of two types of constraints. In the Figure 4-4, there was a required constraint to ensure that the collection class contained a set, ensuring that each Node was unique by having a different key in the key-value pair. The second type of constraint was a transitive closure constraint which ensures a Node is not visited more than once to prevent infinite loops.
There were no common patterns in this work whose list implementation only needed their data structure values read and not modified. The list data structures needed to be modified or be written with a new value. For example, the EXT block bitmap not only required a read operation but also required a write operation to either set or reset its value.
Figure 4-5. List Structures with Nested Data Types

**Multiple lists structure.** A multiple list pattern is shown in Figure 4-6. This pattern consisted of an ordered list and an unordered list, which were used for the directory structure in the FAT file system use-case model and the file system structure in the browser use-case model.
Figure 4-6. Multiple List - Directory Structure

**Pointer-to-lists structure.** This pattern showed the association of multiple tables of different types and defined groupings of associations. This pattern was used to model tables that had rows with lists of different types within the row as shown in Figure 4-7. This pattern grouped different types of lists. Each type of lists could have zero or more instances. An example would be the NTFS MFT. Each row contained a set of attributes, with each attribute being defined by a list of types.

Figure 4-7. Pointers-to-lists Structure
**Detail structure.** This pattern utilized a UML dependency class to document additional information which was associated between the classes on both ends of the association (see Figure 4-8). This pattern was not related in modeling the underlying forensic data structures, but was utilized to provide additional detail between data classes within the model. Figure 4-9 provides an example in xtUML to show how the dependency class was utilized. In the figure there is a dependency class BootDetails on the association which links BootSector and OS. The association class contained the specific model forensic attributes which were read by the OS. The data types for these attributes are identified in Figure 4-10. The actual values of these forensic attributes were not defined in the data type; rather, the attribute data types were defined by the Details type. The Details types defined attributes providing specific information which may be of interest a more technically advanced digital forensic stakeholder.

![Figure 4-8. Detail Structure](image-url)
4.1.1.2 Dynamic patterns (Scripts).

The dynamic patterns in this work were utilized to show the control flow of the scenario. The thread of control was scripted to the specific scenario and was at a very abstract level. The three primary dynamic patterns (shown in Figure 4-11) seen in this work were:

- Case 1: Initial state to final state via a sequential set of intermediated states.
- Case 2: Initial state to a target state and remain in the target state.
• Case 3: Transition from the initial state to a target state back to an initial state.

**Case 1.** Given a start event, the state machine started at an initial state and stepped through a sequence of states until the end state was reached. Each state contained a set of steps (implemented in OAL) that resulted in a corresponding set of attributes to be modified to achieve the desired end-effect of the state, after which there was a transition to the next state.

**Case 2.** Given an event, the initial state transitioned to the appropriate target state. The target state was the final state.

**Case 3.** Given an event, the state machine transitioned to the appropriate target state, executed the requisite set of steps, and transitioned back to the initial state.

**Figure 4-11. Dynamic Patterns**

The Dynamic Pattern implemented per forensic use-case for this work is shown in Table 4-2.
### 4.1.2 Functional grouping definitions.

The functional groups were defined by further analyzing the functional area use-cases based on what was discovered during the development of the specific models. The requisite high level functionality of the underlying computational mechanism was identified along with the relevant data structures. The functional groupings are a key component in defining the top-level models.

**File systems.** Based on the analysis of the scenario, the file system functionality could be further decomposed into four major grouping, the `RetrieveFSMetaData`, `AllocateToDUStorage`, `DetermineStorageLocation`, and `AccessDirectory`. These functions are operating system functions. The `RetrieveFSMetaData` retrieves the file system metadata. The operating system utilizes this information to determine how to interact with the file system, such as determining the location of file system data structures. `AllocateToDUStorage` either reads or allocates contents to memory. The `DetermineStorageLocation` determines the data units to which file contents are to be written. The `AccessDirectory` function accesses the directory to access the meta data of the file. Table 4-3 maps the file system use-case activities/functions to the supporting functional grouping functions.
Table 4-3. File System Use-Case Extension

<table>
<thead>
<tr>
<th>Use-Case Activity/Function</th>
<th>Supporting Functional Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllocateFile</td>
<td>AccessDirectory</td>
</tr>
<tr>
<td>AllocateFile</td>
<td>RetrieveFSMetaData</td>
</tr>
<tr>
<td>AllocateFile</td>
<td>DetermineStorageLocation</td>
</tr>
<tr>
<td>AllocateFile</td>
<td>AllocateToDUSTorage</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>AccessDirectory</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>RetrieveFSMetaData</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>DetermineStorageLocation</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>AllocateToDUSTorage</td>
</tr>
</tbody>
</table>

Table 4-4 shows the mapping of the specific file system forensic data structures being modeled to the functional grouping of this work.

Table 4-4. Key Forensic Data Structures Mapped to Functional Groups

<table>
<thead>
<tr>
<th>Use-Case Activity/Function</th>
<th>FAT</th>
<th>NTFS</th>
<th>EXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetrieveFSMetaData</td>
<td>Boot sector,FSINFO</td>
<td>$Boot</td>
<td>Superblock, group descriptor</td>
</tr>
<tr>
<td>DetermineStorageLocation</td>
<td>FAT</td>
<td>$Bitmap,$STANDARD_INFORMATION,$DATA,$ATTRIBUTE_LIST</td>
<td>Inodes, inode bitmap</td>
</tr>
<tr>
<td>AcessDirectory</td>
<td>Directory Entries</td>
<td>$FILE_NAME,$IDX_ROOT,$IDX_ALLOCATION,$BITMAP</td>
<td>Directory Entries</td>
</tr>
<tr>
<td>AllocateToDUSTorage</td>
<td>Clusters</td>
<td>Clusters</td>
<td>Blocks</td>
</tr>
</tbody>
</table>

Figure 4-12 shows the extended use-case incorporating the additional implementation functions resulting from the data structure groupings that emerged through the analysis. These are operating system functions that are contained in the file system component in the use-case. These functions extend the use-cases.
AllocateFile and DeallocateFile activities/functions and were added using the include relationship shown in the extended use-case.

**Figure 4-12. File System Extended Use-Case**

**Browser.** The decomposed functions for the functional groupings for the browser use-case are, FSStoreRetrieve and DbStoreRetrieve. These functions simply store information gathered by the browser when visiting web sites and downloading files.

The file system representation in the browser model did not need to be modeled to the same level of detail that was required for the file system use-case. The browser model needed only show how the file system could be traversed to gain access to the specified file. Table 4-5 maps the RAM use-case activities/functions to the supporting functional grouping functions.
Table 4-5. Browser Use-Case Extension

<table>
<thead>
<tr>
<th>Use-Case Activity/Function</th>
<th>Supporting Functional Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse</td>
<td>FSStoreRetrieve</td>
</tr>
<tr>
<td>Browse</td>
<td>DbStoreRetrieve</td>
</tr>
<tr>
<td>Download</td>
<td>FSStoreRetrieve</td>
</tr>
<tr>
<td>Download</td>
<td>DbStoreRetrieve</td>
</tr>
<tr>
<td>EnableCookie</td>
<td>FSStoreRetrieve</td>
</tr>
<tr>
<td>EnableCookie</td>
<td>dbStoreRetrieve</td>
</tr>
</tbody>
</table>

Figure 4-13 shows the updated browser scenario use-case, which includes the additional decomposed model functionality.

Figure 4-13. Browser Extended Use-Case
**RAM.** The decomposed functions for the functional groupings for the RAM use-case are, ManageProcesses, CreateSocketConnections, MapSockets, and ManageConnections. ManageProcesses provides the functionality to create processes and to provide the equivalent process information that would be displayed by a standard process list command line. The ManageConnections function provides the equivalent network connection information which would be displayed by a standard network connection list command line. The CreateSocketConnection function allows the application to create a socket network connection. The MapSockets function creates an entry in the process handle table for a socket. This table is utilized to associate a process with a network connection. Table 4-6 maps the RAM use-case activities/functions to the supporting functional grouping functions.

<table>
<thead>
<tr>
<th>Use-Case Activity/Function</th>
<th>Supporting Functional Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListProcesses</td>
<td>ManageProcesses</td>
</tr>
<tr>
<td>StartApplication</td>
<td>ManageProcesses</td>
</tr>
<tr>
<td>CreateNetworkConnection</td>
<td>MapSockets</td>
</tr>
<tr>
<td>CreateNetworkConnection</td>
<td>CreateSocketConnection</td>
</tr>
<tr>
<td>ListNetworkConnections</td>
<td>ManageConnections</td>
</tr>
</tbody>
</table>

The updated scenario diagram, with the additional decomposed functionality, is shown in Figure 4-14. The diagram included the OS_ProcessManager component functionality to address a process life cycle, which included process creation and scheduling. Also, the diagram included the OS_IOManager component functionality to address network communication.
Figure 4-14. RAM Extended Use-Case
4.2 Top-level model & profile constructive analysis (steps 5-7).

The following sections identify the requisite analysis and validation in determining the top-level models and the associated profiles.

4.2.1 File systems

The top-level model for file systems was based on the file system functional groupings. Table 4-7 maps the functional groupings for the top-level model and the implementation data structures patterns used to implement the top-level model.

The file system top-level model is shown in Figure 4-15. The associated data structures from the top-level model are identified in Figure 4-16. The DataUnitManager is a collection class of DataUnit, which models disk storage. The Directory is a combination of lists used to model the directory structure and is also a collection class of DirectoryEntry. DirectoryEntry contains file information along with references to the appropriate set of DataUnit. The FSMetaStore contains the file system’s metadata. The OS contains the state machine.

<table>
<thead>
<tr>
<th>Supporting Function/Component</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetrieveFSMetaData</td>
<td>SUEN</td>
</tr>
<tr>
<td>DetermineStorageLocation</td>
<td>DOLM</td>
</tr>
<tr>
<td>AccessDirectory</td>
<td>DOTMDUTM</td>
</tr>
<tr>
<td>AllocateToDUDStorage</td>
<td>DULM</td>
</tr>
<tr>
<td>Application</td>
<td>SUEN</td>
</tr>
<tr>
<td>OS</td>
<td>SUEM</td>
</tr>
</tbody>
</table>
Figure 4-15. File System Top-Level Model
Figure 4-16. File System Top-level Model Data Types
The operating system dictated the control flow for the file system scenario. The behavioral states and definitions are shown in Table 4-8. The state machine is shown in Figure 4-17. The states represented the set of steps to perform the scenario based on a file “save as” event or a “delete file” event. The “save as” event was initiated at the word processing application. The delete event was the result of moving the file to trash. In the file system scenario, the state machine went through a sequential set of steps for allocation. Based on an operator-initiated delete, the state machine also traversed through a sequential set of states. These states were the same across all the implementations, however, the ordering of the states differed.

Table 4-8. File System States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Initialize scenario</td>
</tr>
<tr>
<td>BootStructDataDetermined</td>
<td>File System Meta Data is read by operation system</td>
</tr>
<tr>
<td>FileLocDetermined</td>
<td>Determine location of target directory and file location within the directory</td>
</tr>
<tr>
<td>FileEntryCreated</td>
<td>Create initial file entry structure and/or fill in initial information for the file entry</td>
</tr>
<tr>
<td>WriteContents</td>
<td>Determine location to write contents/write consents</td>
</tr>
<tr>
<td>FileSaveComplete</td>
<td>Intermediate state between actions</td>
</tr>
<tr>
<td>DeleteInitialized</td>
<td>Initialize data structures for delete</td>
</tr>
<tr>
<td>DirectoryEntryCleared</td>
<td>Clear/Unallocated file structures in target directory</td>
</tr>
<tr>
<td>AllocationStructuresCleared</td>
<td>Clear/Unallocated file allocation data structures</td>
</tr>
<tr>
<td>Complete</td>
<td>End State</td>
</tr>
</tbody>
</table>
Figure 4-17. File System Top-Level State Machine
The key profile elements for the file system static profile are identified in Table 4-9. The static profile was derived from the top-level static model and is shown in Figure 4-18. The profile stereotypes were consistent with the top-level classes which represented the forensic data structures. In addition, the data types that were consistent with the forensic attributes were utilized to define the data types of the tags for the stereotypes. The tag data types (see Figure 4-19) were categorized as related to either evidence, operating system, and applications. In addition, a time data type is identified along with enumerated types for file system identification. Constraints were added to ensure unique elements in collections and to address transitive closure to prevent infinite loops. The dynamic profile for the states of the state machine is depicted in Figure 4-20.
<table>
<thead>
<tr>
<th>Top-level Element</th>
<th>Top-level Type</th>
<th>Profile Element</th>
<th>Stereotype, Data Type, or Tag</th>
<th>Profile Meta Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Class</td>
<td>OS</td>
<td>Stereotype</td>
<td>Class</td>
<td>Operating System.</td>
</tr>
<tr>
<td>Application</td>
<td>Class</td>
<td>Application</td>
<td>Stereotype</td>
<td>Class</td>
<td>Application whose content is stored in a file.</td>
</tr>
<tr>
<td>FSMetaStore</td>
<td>Class</td>
<td>FSMetaStore</td>
<td>Stereotype</td>
<td>Class</td>
<td>Contains the file system meta data.</td>
</tr>
<tr>
<td>Directory</td>
<td>Class</td>
<td>DirectoryRoot</td>
<td>Stereotype</td>
<td>Class</td>
<td>Contains other directories or directory entries. Top level directory is the “root”.</td>
</tr>
<tr>
<td>DirectoryEntry</td>
<td>Class</td>
<td>DirectoryEntry</td>
<td>Stereotype</td>
<td>Class</td>
<td>Contains the meta information for the files or subdirectories.</td>
</tr>
<tr>
<td>FileDataUnitRef</td>
<td>Class</td>
<td>FileDataUnitRef</td>
<td>Stereotype</td>
<td>Class</td>
<td>A set of references to DataUnit(s) where the file content is written.</td>
</tr>
<tr>
<td>DUAllocationMechanism</td>
<td>Class</td>
<td>DUAllocationMechanism</td>
<td>Stereotype</td>
<td>Class</td>
<td>Data structure which keeps track of available data units.</td>
</tr>
<tr>
<td>AllocationCell</td>
<td>Class</td>
<td>AllocationCells</td>
<td>Stereotype</td>
<td>Class</td>
<td>An individual cell/element in the DUAllocationMechanism.</td>
</tr>
<tr>
<td>DataUnitMangement</td>
<td>Class</td>
<td>DataUnitMangement</td>
<td>Stereotype</td>
<td>Class</td>
<td>The mechanism which manages the access to the file system data units.</td>
</tr>
<tr>
<td>DataUnit</td>
<td>Class</td>
<td>DataUnit</td>
<td>Stereotype</td>
<td>Class</td>
<td>An individual data unit which can be written to. Typically a cluster or block.</td>
</tr>
</tbody>
</table>
Figure 4-18. File System Static Profile
Figure 4-19. File System Profile Tag Types
Figure 4-20. File System State Behavior Profile
4.2.2 Browsers

The browser scenario was at a higher level of abstraction than the file system use-case. The same model was used across all three implementations; the differences were with respect to the ways in which the scenarios were initially configured. With regard to the file system, file paths and files names were different across implementations. Additionally, databases, tables, and table contents had different configurations between implementations.

The mapping of the browser functional groups and the pattern implementation is shown in Table 4-10.

<table>
<thead>
<tr>
<th>Supporting Function/Component</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbStoreRetrieve/ DataBase</td>
<td>Unique</td>
</tr>
<tr>
<td>FSSstoreRetrieve</td>
<td>DOTMDUTM (Directory)</td>
</tr>
<tr>
<td>Browser</td>
<td>SUEM</td>
</tr>
<tr>
<td>OS</td>
<td>SUEM</td>
</tr>
</tbody>
</table>

The browser top-level class diagram is shown in Figure 4-21. The associated data types for the top-level diagram are depicted in Figure 4-22. The major classes in the class model were the WebBrowser, the Database, the OS, and the classes representing the files system (Directory and File) that were collection classes. The Downloads, History, and Cookie data types are consistent with the forensic attributes. The WebBrowser application contained the state machine. The browser state machine is shown in Figure 4-23.
Figure 4-21. Browser Top-Level Model
Figure 4-22. Browser Top-Level Model Data Types
The state machine states and their descriptions are in Table 4-11. The application state changed based on one of three transition events:

- Download: An artifact has been downloaded.
- History update: Change in URL.
- Cookie update: Cooking information is stored.

After the transition change has been completed the application goes back to its typical state, ActionProcessed.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionProcessed</td>
<td>This is the typical state, waiting to process a user request</td>
</tr>
<tr>
<td>HistoryUpdated</td>
<td>When a URL is visited the history is updated</td>
</tr>
<tr>
<td>CookieUpdated</td>
<td>When cookies are enabled and a site is visited which utilizes cookies, cooking information is updated</td>
</tr>
<tr>
<td>DownloadUpdated</td>
<td>When a user performs download, the download information is updated</td>
</tr>
</tbody>
</table>
Figure 4-23. Browser State Machine
The key profile elements for the Browser profile are identified in Table 4-12. The static profile was derived from the Browser top-level static model and is shown in Figure 4-24. The profile stereotypes were consistent with the top-level classes that represented the forensic data structures. The profile also identified OCL constraints. In addition, the data types that reflected the forensic attributes were utilized to define the data types of the tags for the stereotypes. The tag data types, shown in Figure 4-25, were categorized as related to either evidence, operating system, or application. In addition, data types for time and an enumeration for browser types were identified. Constraints were added to ensure unique elements in collections and to address transitive closure to prevent infinite loops. The dynamic profile for the behavior states is depicted in Figure 4-26.
<table>
<thead>
<tr>
<th>Top-level Element</th>
<th>Top-level Type</th>
<th>Profile Element</th>
<th>Stereotype, Data Type, or Tag</th>
<th>Profile Meta Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>Class</td>
<td>Content</td>
<td>Stereotype</td>
<td>Class</td>
<td>Content of file.</td>
</tr>
<tr>
<td>Browser</td>
<td>Class</td>
<td>Browser</td>
<td>Stereotype</td>
<td>Class</td>
<td>Web browser application.</td>
</tr>
<tr>
<td>OS</td>
<td>Class</td>
<td>OS</td>
<td>Stereotype</td>
<td>Class</td>
<td>Operating System.</td>
</tr>
<tr>
<td>Database</td>
<td>Class</td>
<td>Database</td>
<td>Stereotype</td>
<td>Class</td>
<td>Database. Contains Tables.</td>
</tr>
<tr>
<td>Table</td>
<td>Class</td>
<td>Table</td>
<td>Stereotype</td>
<td>Class</td>
<td>Contains Row(s).</td>
</tr>
<tr>
<td>Row</td>
<td>Class</td>
<td>Row</td>
<td>Stereotype</td>
<td>Class</td>
<td>Contain data and has meta data on row.</td>
</tr>
</tbody>
</table>
Figure 4-24. Browser Use-Case Profile
Figure 4-25. Browser Tag Data Types
Figure 4-26. Browser State Behavior Profile
4.2.3 RAM

The RAM scenario was at a higher level of abstraction than the file system use-case. Due to the lack of available information for implementation specifics, the class and state machines were a high level, generic, abstract, textbook-configuration that represented all the configurations. The top-level dynamic and static models for RAM were the same as the implementation specific models. The differences between the models were in the terminology used to describe the specifics of the process and network evidence. These differences were reflected in the data types. The patterns associated with the functional areas, applications, and OS are shown in Table 4-13.
### Table 4-13. RAM Functional Group/Components to Patterns

<table>
<thead>
<tr>
<th>Supporting Function/Component</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManageProcesses</td>
<td>DOLM</td>
</tr>
<tr>
<td>CreateSocketConnection</td>
<td>DULM</td>
</tr>
<tr>
<td>ManageConnections</td>
<td>DULM</td>
</tr>
<tr>
<td>MapSockets</td>
<td>DULM</td>
</tr>
<tr>
<td>CommandLine</td>
<td>SUEN</td>
</tr>
<tr>
<td>Application</td>
<td>SUEM</td>
</tr>
<tr>
<td>OS</td>
<td>SUEM</td>
</tr>
</tbody>
</table>

The RAM class diagram for the top-level model is shown in Figure 4-27. The association data types for the RAM top-level model are identified in Figure 4-28 and represent the key forensic attributes. The OS contained the state machine and dictated the control flow. The state machine for the OS is shown in Figure 4-29 and the state machine for the Application is shown in Figure 4-30. The viewing tool modeled the output of the OS commands that provided either a list of running processes or the open connections. The ProcessManagement class was a collection of processes executing on the machine. The collection process (i.e., process tree) was implemented as an ordered collection. The process had an association with a FileDescriptorTable (i.e., handle table) that was a collection of available handles. In this case, the model represented socket file descriptors. The Socket was associated with a CommunicationsObject which is a member of the ConnectionManagement collection class. The application itself was associated with the OS and if it was a network enabled application it was associated with Socket. The OS was in a SteadyState unless there was an action:

- to add a process (e.g., application starting up), or
• to add a socket (e.g., application establishes a network connection).

These actions occurred when a new application instance was created. The application had a state machine, and depending on the initial startup parameters, was either in a network enabled state or not. The OS states and descriptions are listed in Table 4-14 and the application state machine states and descriptions are listed in Table 4-15.

The run of the scenario set up various instances of an applications and modeled the output of the viewing tool. The system process list or open connection list was shown by executing the View Tool, which was representative of command lines for processes and network connections.
Figure 4-27. RAM Top-Level Model
Figure 4-28. RAM Top-level Model Data Types

<table>
<thead>
<tr>
<th>State (OS)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SteadyState</td>
<td>Operating system waiting state</td>
</tr>
<tr>
<td>ProcessAdded</td>
<td>Process is added to process list</td>
</tr>
<tr>
<td>SocketAdded</td>
<td>Process is added to process list and a socket handle is created</td>
</tr>
</tbody>
</table>

Figure 4-29. RAM OS State Machine
Table 4-15. RAM Application States

<table>
<thead>
<tr>
<th>State (Application)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotInit</td>
<td>Application not initialized</td>
</tr>
<tr>
<td>Executing</td>
<td>Application is initialized and is not a network enabled</td>
</tr>
<tr>
<td>ExecutingAsNetworkClient</td>
<td>Application is initialized and is network enabled</td>
</tr>
</tbody>
</table>

Figure 4-30. RAM Application State Machine

The key profile elements for the RAM profile are identified in Table 4-16. The static profile was derived from the RAM top-level static model and is shown in Figure 4-31. The profile stereotypes were consistent with the top-level classes which represented the forensic data structures. In addition, the data types that reflected the forensic attributes were utilized to define the data types of the tags for the stereotypes. The tag data types, shown in Figure 4-32, were categorized as related to either evidence, operating system, or application. In addition, data types for time and an enumeration for Application types were identified. Constraints were added to ensure unique elements in collections and to address transitive closure to prevent infinite loops. The behavior profile for the OS states is depicted in Figure 4-33 and the behavioral profile for the Application behavior states is depicted in Figure 4-34.
<table>
<thead>
<tr>
<th>Top-level Element</th>
<th>Top-level Type</th>
<th>Profile Element</th>
<th>Stereotype, Data Type, or Tag</th>
<th>Profile Meta Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Class</td>
<td>Application</td>
<td>Stereotype</td>
<td>Class</td>
<td>Application whose process and network connections are of interest.</td>
</tr>
<tr>
<td>OS</td>
<td>Class</td>
<td>OS</td>
<td>Stereotype</td>
<td>Class</td>
<td>Operating System</td>
</tr>
<tr>
<td>ViewingTool</td>
<td>Class</td>
<td>CommandLineTool</td>
<td>Stereotype</td>
<td>Class</td>
<td>Command line tool to either get process list or open connections.</td>
</tr>
<tr>
<td>Socket</td>
<td>Class</td>
<td>Socket</td>
<td>Stereotype</td>
<td>Class</td>
<td>Provides API for application to receive and send information.</td>
</tr>
<tr>
<td>CommsSocketObject</td>
<td>Class</td>
<td>CommsSocketObject</td>
<td>Stereotype</td>
<td>Class</td>
<td>Represents the network stack between the application Sockets to the network “wire”.</td>
</tr>
<tr>
<td>ConnectionManagement</td>
<td>Class</td>
<td>ConnectionManagement</td>
<td>Stereotype</td>
<td>Class</td>
<td>Maintains a list of network connections.</td>
</tr>
<tr>
<td>ProcessManagement</td>
<td>Class</td>
<td>ProcessManagement</td>
<td>Stereotype</td>
<td>Class</td>
<td>Maintains the list of processes.</td>
</tr>
<tr>
<td>Process</td>
<td>Class</td>
<td>Process</td>
<td>Stereotype</td>
<td>Class</td>
<td>An individual processes. Contains the forensic information of interest.</td>
</tr>
<tr>
<td>FileDescriptorTable</td>
<td>Class</td>
<td>FileDescriptorTable</td>
<td>Stereotype</td>
<td>Class</td>
<td>Contains the network handles for the process of interest.</td>
</tr>
<tr>
<td>FileDescriptor</td>
<td>Class</td>
<td>FileDescriptor</td>
<td>Stereotype</td>
<td>Class</td>
<td>Individual file handle for a network object.</td>
</tr>
</tbody>
</table>
Figure 4-31. RAM Static Profile
Figure 4-32. RAM Profile Tag Data Types
Figure 4-33. RAM OS State Profile

Figure 4-34. RAM Application State Profile
4.2.4 Model equivalence

The analysis determined that the top-level models were equivalent to the implementation specific models in all three forensic areas of this work.

**File systems.** The file system top-level model was equivalent to the file system implementation specific models. The results for the model equivalent scripts are shown in Table 4-17 for the file system area. The only significant anomalies were due to the differences in when time was recorded across the file systems, specifically with regard to the type of time stamps (e.g., modify, access, create, write) and where the time stamps were located (e.g., at the file entry level or the parent directory level). For the implementation differences, the forensic attribute for time was at a too low level of abstraction. If the time attribute was deemed to be of significant interest, it would need to be re-evaluated and the appropriate level of abstraction would need to be defined. For instance, perhaps the time on file updates might be of interest.

There were also some minor difference in the order in which the forensic attributes were generated, however this had no effect on the forensic attribute values.
Table 4-17. File System Equivalence

<table>
<thead>
<tr>
<th>Specific Model</th>
<th>Common Steps with Top-Level Model</th>
<th>Total Steps</th>
<th>%</th>
<th>Step Delta Comments</th>
<th>Equivalent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAT</td>
<td></td>
<td>30</td>
<td>38</td>
<td>78</td>
<td>Due to how FS implements times</td>
</tr>
<tr>
<td>EXT</td>
<td></td>
<td>31</td>
<td>37</td>
<td>82</td>
<td>Due to how FS implements times</td>
</tr>
<tr>
<td>NTFS</td>
<td></td>
<td>34</td>
<td>38</td>
<td>89</td>
<td>Due to how FS implements times</td>
</tr>
</tbody>
</table>

**Browsers.** The browser top-level model was equivalent to the browser implementation specific models. Since the specific models and top-level models were the same, the differences in the model were in the scenario configuration of the file system and data base models in the browser implementation.

There were a few instances where the location of these attributes could not be found in the existing documentation. It was verified that the relevant information does exist in the browser, however, the available documentation in the proprietary implementation was not readily available. In addition, there was slight variance in how the implementations stored the information as variables in the data base. For example, in the download forensic attributes, for the name of the downloaded document, one browser included the path in its attribute and another implementation used separate variables for name and for path. Overall, there was an approximately 94% mapping between each of the specific implementation attributes and that of the browser forensic attributes.
**RAM.** The RAM top-level model was equivalent to the RAM implementation specific models. Since the specific models and top-level models were the same the differences in the model were in the scenario configurations with regard to the process attributes and network attributes. As with the browsers, the documentation for the specific proprietary implementations was incomplete. However, there was only one instance in which the attribute name could not be determined. Overall, there was an approximately 95% mapping between each of the specific implementation attributes and that of the RAM forensic attributes.

4.3 Digital forensic profile constructive analysis (step 8)

The static digital forensic profile was constructed by integrating the forensic use-case profiles. Since the behavior profiles were use-case specific they did not change from the area behavior profiles and are not addressed in this section. The description of the profile elements are provided in Appendix A.

The resulting profile packages are depicted in Figure 4-35 and further described in Table 4-18.

![Figure 4-35. DF Profile Package View](image-url)
Table 4-18. DF Profile Package Description

<table>
<thead>
<tr>
<th>Profile Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFOS</td>
<td>Operating system stereotypes.</td>
</tr>
<tr>
<td>DFFFileSystem</td>
<td>File system stereotypes.</td>
</tr>
<tr>
<td>DFApplication</td>
<td>Application stereotypes.</td>
</tr>
<tr>
<td>DFRAM</td>
<td>RAM stereotypes.</td>
</tr>
<tr>
<td>DFData Types</td>
<td>Data types supporting profile stereotypes.</td>
</tr>
<tr>
<td>DFEvidenceTypes</td>
<td>Evidence data types supporting profile stereotypes.</td>
</tr>
</tbody>
</table>

The operating systems were common across the categories. As a result, the OS stereotypes were transferred from the area profiles to a new OS profile as shown in Figure 4-36. The OS profile contains all of the OS elements that were required by the area forensic profiles. In this profile the OS is composed of:

- ProcessManager: partially equivalent to the OS in the RAM.
- IOManager: partially equivalent to the OS in the RAM.
- FileSystemManagerDetailed: equivalent to the OS in the File System Profile.
- FileSystemManager: equivalent to the OS in the Browser Profile.

To show that stereotypes can extend other types of UML meta-classes, OsUserCall, OsKernalCall, and ApiCall stereotypes extending the Association meta-class are included in the OS profile.

Since the browser itself is an application, the Browser profile was the basis for the Application profile. This was accomplished by incorporating stereotypes from both the File System and RAM area profiles in the modeled application. The various application stereotypes were inherited from the Application class (see Figure 4-37).
The DF File System Profile, shown in Figure 4-38 is similar to the associated File system area profile, without the operating system and application stereotypes. Likewise, the DF RAM Profile, Figure 4-39, is similar to RAM area profile, without the operating system and applications stereotypes.

Lastly, the profiles DF Evidence Types, shown in Figure 4-40, are directly related to the forensic attributes. The DF Supporting Types, shown in Figure 4-41, provided the data types for application types, OS, and specific data types for the detailed file system model. In addition, a DFTime data type was identified with enumerations to be used in identifying specific configurations. It should be noted that OCL constraints were included to:

- Ensure that collections elements had unique keys.
- Transitive closure to prevent infinite loops.
- Time attributes ranges were defined.
Figure 4-36. DF OS Profile
Figure 4-37. DF Application Profile
Figure 4-38. DF File System Profile
Figure 4-39. DF RAM Profile
Figure 4-40. DF Evidence Types
Figure 4-41. DF Types
4.4 Metric results

Figure 4-42 identifies the TLAM results. The file system top-level model resulted in the greatest reduction in TLAM case 1. Both the browser and RAM top-level models had a reduction of 3 to 1. The second TLAM case shows that there is a 0 percent reduction of elements when the browser and RAM top-level models are compared to each of their respected implementations. In the browser and RAM use-cases, the model implementations and top-level models are the same, therefore there is no reduction seen for TALM case 2 and the minimum reduction ratio is seen for TLAM case 1. On the other hand, the file system top-level model shows significant reductions in elements when compared to the NTFS and EXT implementations. This was reflective in that the complexity of the NTFS and EXT implementations are significantly greater than the complexity of the top-level model equivalent. The results indicated that for implementations which were 100% common were at a maximum level of abstraction.
Figure 4-42. TLAM Results

Figure 4-43 identifies the metrics results for ICM. The set of common patterns were defined as the patterns utilized in at least two of the use-cases. ICM case 1 shows that all the implementations had a relatively high degree of common pattern usage. The browser and RAM use-cases had no variation for the ICM metric. This was due to the implementations being identical. The file system implementations showed a variation in the metric since their implementations vary.

ICM case 2 shows that the browser and RAM use-cases have a value of 1 showing maximum commonality, which was as expected. On the other hand the file system use-case shows less than 100% commonality. Investigating the file system use-case further, the ICM was applied to three aspects of the file system use-case implementations, the application, the operating system, and the functional groups. The calculation for these aspects were:
The operating system and applications were common across implementations. Further investigation of applying ICM case 2 to the functional groups is shown in Figure 4-44. The Data Unit Allocation functional group was relatively simple functionality representing the data units of the file system. This group was simply modeled as a collection across all the implementations. Other than naming (e.g. blocks, clusters), the model implementation of this simple functionality is common across the implementations. There were also similarities in the FS Meta Store functional group that also has relatively simple functionality in that primarily functions as a store. However, there was a significant differences in implementations in the Data Unit Allocation and Directory functional groups which were reflected in the resulting ICM case 2 calculation.
The results for ICM case 3 is shown in Figure 4-45. The results show that there is commonality between the use-cases.
Figure 4-45. ICM Case 3 (Across Use-Case) Results

**Metrics summary/conclusions.**

The file system use-case top-level model showed significant reduction of elements with the abstracted top-level model when compared to the detailed models. However, this was not the case with the browser or RAM use-cases since the implementations of the specific models were the same as the top-level model.

Commonality was seen throughout the implementations within the forensic areas and across implementations. This commonality varied significantly but increased as the level of abstraction increased. The metrics results were supportive that there was commonality across the models which was consistent with what was expected by inspection of the implementations.

An important observation was that a higher degree of commonality between the implementations corresponded to a smaller reduction in the number of elements in the top-level model when compared against the specific model implementations. It was also observed that less commonality between implementations was an indicator of increased complexity across the use-case. This is suggestive that there is an inverse relationship between commonality and complexity, as measured by the level of abstraction between the top-level model and implementation models. Another significant observation is that the commonality in model implementations is
suggestive that there is commonality in the actual computational mechanisms which were the subject of the model.

4.5 Constructive procedure for model extension

The constructive method to extend top-level models and/or extend the digital forensic profile is outlined in Figure 4-46.

<table>
<thead>
<tr>
<th>Construction Method For Top-Level General Model and Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1: Define Model Objectives</strong></td>
</tr>
<tr>
<td>• Determine forensic use-case diagram.</td>
</tr>
<tr>
<td>• Define use-case Scenario.</td>
</tr>
<tr>
<td>• Identify the level of abstraction by identifying the forensic attributes.</td>
</tr>
<tr>
<td>• Identify implementations to model.</td>
</tr>
<tr>
<td><strong>Step 2: Develop Specific Models</strong></td>
</tr>
<tr>
<td>• Implement specific models.</td>
</tr>
<tr>
<td><strong>Step 3: Execute Models</strong></td>
</tr>
<tr>
<td>• Execute Models and capture model behavior.</td>
</tr>
<tr>
<td>• Verify behavior model with expect results.</td>
</tr>
<tr>
<td><strong>Step 4: Discover</strong></td>
</tr>
<tr>
<td>• Discover and define concrete implementation patterns and functional groups.</td>
</tr>
<tr>
<td>• Extend use-case with functional grouping.</td>
</tr>
<tr>
<td><strong>Step 5: Develop Top-Level Model</strong></td>
</tr>
<tr>
<td>• Utilize the functional groups to develop the top level model.</td>
</tr>
<tr>
<td>• Map the implementation patterns into the functional groupings.</td>
</tr>
<tr>
<td><strong>Step 6: Validate Top-Level Model</strong></td>
</tr>
<tr>
<td>• Demonstrate specific models can be transformed to general model.</td>
</tr>
<tr>
<td><strong>Step 7: Create Use-Case Profile</strong></td>
</tr>
<tr>
<td>• Create use-case profile from use-case top-level model.</td>
</tr>
<tr>
<td><strong>Step 8: Define/Update Digital Forensic Profile</strong></td>
</tr>
<tr>
<td>• Create digital forensic profile or update existing profile</td>
</tr>
</tbody>
</table>

Figure 4-46. Constructive Method

This work has demonstrated this constructive method was repeatable by developing the top-level models, the profiles, and the updated use-case diagrams for the three scenarios on which this work was focused.
There are three possible cases in which this constructive method would be utilized:

Case 1: Adding implementations to an already existing use-case.

Case 2: Abstraction level changes.

Case 3: Adding a new forensic use-case.

**Case 1.** Adding new implementations may result in further refinement of the top-level model and possible modifications to the profiles. The new implementation would have been developed utilizing the existing modeling objectives and would start on step 2. After implementation, the transformation script for the new model implementation would need to be developed, executed and used to validate model equivalency.

**Case 2.** A change in abstraction would have required a change in the use-case scenario and forensic attributes starting at step 1. This may also result in an alternative use-case top-level model and possible modifications to the profiles.

**Case 3.** A new use-case would result in all the steps to be exercised. A new top-level model and use-case profile would result along with potential changes to the overarching forensic profile.

4.6 Application analysis

**Visual animation description.** The GUI depicted the application, operating system, and the various file system data structures, which included, directories, FAT, boot sectors, and clusters. The display, without detailed information, is shown in
Figure 4-47. The additional GUI details which provide the higher fidelity for a more advanced stakeholder is shown in Figure 4-48.

The animation showed the component that determined the thread of control and the data structure which was the focus of the thread of control for a given time. The thread of control was typically an application or operating system component. The initial conditions were set for the relevant attributes of the directory, FAT, cluster, application, and operating system model elements. The initial state was configured by the initial configuration scripts. There were four different scenarios demonstrated for the use-case. The initial configuration varied for each of the scenarios; the other aspects of the model (e.g., class diagrams, state machine diagrams, OAL, and the application source code) required no change. During the animation, the user selected the “Next” button to go to the next step of the animation. The next animation step occurred when a forensic attribute of the data structure changed, when the source of the thread of control changed, or when the target forensic data structure changed. The method chosen to show the next step was to show an arrow between the source and target components. The top-level description of the next step action was on the link.

A more detailed description of the next step action was provided at the top of the display. These detailed descriptions utilized the detailed model pattern previously identified. For stakeholders requiring additional information, the following display options were available:

- **Details**: Provided the underlying detailed information for a particular forensic attribute, such as a top-level description and the bit position.
- **Location (Loc)**: Provided the cluster location of the data structure.
• Notes: Provided additional descriptive information, such as, column headers of the Directories.

• Evaluation (Eval): Provided additional information, such as slack.
Figure 4-47. Application View (No Details)
Figure 4-48. Application View (Detailed)
**Expert review.** It should be noted that the examiners were exposed only to the application and not the other aspects of this work. As they observed a demonstration of the application they were asked to comment on the following:

- Would this type of animated application be useful (as illustrated by the FAT file allocation/deallocation) from an educator/student or analyst stakeholder role?
- Would this type of animated application be useful for other digital forensic stakeholder roles?

The comments provided by the expert reviewers are summarized in Table 4-19. Comments were received on following topics: domain complexity, level of detail, model sources, roles, and how to improve the tool.
<table>
<thead>
<tr>
<th>Comment Topic</th>
<th>Comment (Paraphrased)</th>
<th>Reviewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Complexity</td>
<td>There are uncountable implementations. Even with the file system, there are a lot of versions.</td>
<td>2</td>
</tr>
<tr>
<td>Level of Details</td>
<td>Operating system routines may be doing additional actions which may be of interest.</td>
<td>2</td>
</tr>
<tr>
<td>Model Sources</td>
<td>Implementation details are often not known, the only insight is what is published via research or by reverse engineering.</td>
<td>2</td>
</tr>
<tr>
<td>Model Sources</td>
<td>What authoritative source is a model based on? Why should the model be trusted?</td>
<td>2</td>
</tr>
<tr>
<td>Roles</td>
<td>For the Student/Educator role, could be used to enhance teaching.</td>
<td>1</td>
</tr>
<tr>
<td>Roles</td>
<td>Could be used to help remind/refresh analyst on how to do a procedure.</td>
<td>1</td>
</tr>
<tr>
<td>Roles</td>
<td>Judge/Jury: There is too much detail in the application for a judge and jury</td>
<td>2</td>
</tr>
<tr>
<td>Roles</td>
<td>Trained analyst already has accepted forensic tools.</td>
<td>2</td>
</tr>
<tr>
<td>Roles</td>
<td>Did not see how would be useful for analysis, but good for teaching.</td>
<td>1</td>
</tr>
<tr>
<td>Application Improvement</td>
<td>Adjustments to displays recommended to facilitate in understandability (running list of previous steps, &quot;not a fan of the arrows&quot;).</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4-19. Expert Review Comments
Analysis of comments.

**Complexity/Detail.** The models can be extended to address the particular implementation of interested utilizing the constructive method. Extensibility was one of the objectives of this work. The extensibility is discussed in terms of breadth, the ability to add new implementations or new versions of extending implementations, and in terms of depth, adding additional model fidelity. For example, extensibility was shown by modeling three implementations per scenario. Modeling to various degrees of detail was demonstrated with the relatively low level modeling of the file system area as opposed to the very high granularity modeled for the RAM area.

**Sources.** This work utilized sources that are generally used in an educational setting or are publically available. For the purpose of this work these sources were adequate to demonstrate the utility of modeling. Nevertheless, the model could easily be adjusted to include other sources. For example, one reviewer provided more authoritative sources for the file system area. The use of these sources would have resulted in minor changes to the data type naming, but would have had no overall effect on the focus of this work. However, it should be noted that in modeling efforts, for the model to be trusted, it is important to obtain agreement from the model stakeholder on the set of sources on which the model is to be based.

**Roles.** The indication that this type of tool could be beneficial to facilitate understandability in digital forensics was encouraging. The focus on the types of stakeholders to whom this work would be useful was limited to the educational role, possibly due to the lack of functionality of the demonstrated application. Perhaps
showing different views or providing more advanced visualizations would have provided additional insight into how modeling could be beneficial for the analysis role or other forensic stakeholder roles. In addition, models can be developed to support other application types. For example, models can be queried and used as a database to retrieve configuration information or could be used with applications which are based on artificial intelligence (AI) engines to make inferences or to find patterns in collected data.

**Application improvement.** The recommended improvements would be useful from both an educational perspective and potentially to assist an expert analyst.

In sum, it was found that modeling could be used to develop applications that would be of use to an educator. The feedback from experts in the field was mixed about whether the tool would assist those in other forensic roles.

### 4.7 Findings summary

The overall findings address what was found during implementation, test, analysis and expert review. The associated finding for the objectives are as follows:

**Facilitate learning and analysis application.** An application was developed to provide an animation utilizing artifacts generated by the model. The detail pattern was utilized to provide additional detail for the more technically advanced stakeholders.

An issue that came up during the expert review, which was also encountered in the development of the specific models, was determining the authenticated sources to
utilize. The availability of the details on the devices and components to be modeled were limited. From a stakeholder perspective, if the correct authenticated sources are not utilized, the stakeholder may not have trust in the model. The root cause of the issue of limited source information is that when new implementations are introduced to the marketplace, a significant amount of research to identify what is important from a forensic perspective is required. In one extreme case, the implementations are proprietary and the relevant details need to be determined. In these cases, research is performed to reverse engineer the design so that forensic techniques can be developed. In the other extreme case, the source code is available and therefore everything is known. However, this is at a much lower level of abstraction that needs to be understood, which requires significant research to identify what is important from a forensic perspective. Regardless, research needs to be conducted before a model can be created.

The expert review identified that the animation application could be useful in an educational setting. The approach was not as positively received for an analyst stakeholder roles, but there were some thoughts on how the approach may be useful to the analyst. There were also concerns with regard to other stakeholder roles which would need to be addressed.

It can be concluded that modeling can have the potential to facilitate applications to enhance digital forensic understanding. The question that this work did not fully satisfy was that multiple stakeholder roles could be addressed by providing different abstraction levels.
Reduced complexity. The four findings which relate to the complexity objective for the constructive method are as follows:

- There were nine static patterns and three dynamic patterns found.
- Top-level models were developed for the three use-cases.
- Model equivalence was shown utilizing transformation.
- Metrics provided model commonality and model abstraction measures.

The discovery of model implementation patterns was an indicator that there are commonalities across actual implementations. The development of the top-level models demonstrated that a top-level abstract model could be used to describe different implementations and to reduce complexity. It was shown that the implementation specific models could be transformed to the top-level model. These results provided an indicator that an abstracted top-level model can be constructed to describe different unique implementations by abstracting out how the implementations are common.

The metrics suggest that there is an inverse relationship between commonality and complexity, as measured by the level of abstraction between the top-level model and implementation models. In addition, the commonality in model implementations which was quantified by the metrics, is suggestive that there is commonality in the actual computational mechanisms which were the subject of the models.

The degree of abstraction increased as the level of commonality increased. This is seen with browser and RAM use-cases which were modeled at a higher level of
abstraction than the file system model and substantiated with the TLAM metric. There were commonalities identified across model implementations within use-cases and across use-cases. The ICM provided a relative measure to identify additional areas in which to investigate potential common functions across implementations.

**Extensibility.** Extensibility of the constructive method was verified in that its steps were repeatable. Extensibility of a top-level model was shown within each use-case through the addition of three implementations. The addition of three use-cases demonstrated the extensibility of the digital forensic profile.

**DF UML profile.** A digital forensic profile was constructed from this work. The elements of the profile can be traced to implementation details. The profile was systematically constructed from top-level models that were constructed from implementation-specific executable models.
CHAPTER 5

5 CONCLUSION

5.1 Summary

Digital forensics is complex in that there are a combinatorially explosive number of types and configurations of digital devices and the components of which they are composed. In addition, there is a wide range of technical skill required of digital forensic stakeholders, depending on their roles in their organizations. This work utilized software engineering techniques based on executable models to identify two approaches to manage these digital forensic complexities.

The first approach was to manage the complexity by identifying commonalities across implementations to provide a single abstracted view of these implementations in a top-level executable model. The top-level model was utilized to develop three use-case profiles. The three use-case profiles were integrated to construct the digital forensic profile. The digital forensic profile would extend software modeling terminology to terminology that is meaningful to individual digital forensic stakeholders. In addition, a method was identified and demonstrated how the top-level models and the digital forensic profile could be further extended. Another result from this approach was the introduction of new metrics to quantify model abstraction and to identify commonalities between implementations. The second approach was to utilize artifacts generated from the execution of the model to provide a visualization application of evidence creation. This visualization would help the targeted stakeholder to better understand the complexities of a particular digital forensic scenario. These two approaches could be utilized to facilitate the application of software engineering-based modeling holistically across the digital forensic domain.
5.2 Conclusion

As a result of this work, the initial questions can be addressed. There is commonality across actual device and component implementations. That is, there is utility in applying software engineering modeling techniques to digital forensics for managing complexity and promoting understandability. In addition, the potential exists to develop an approach applying software modeling holistically across the digital forensic domain. This conclusion is based on all the stated objectives of this work:

1. Addresses domain complexity by utilizing model artifacts to facilitate the development of applications for learning and analysis which support multiple digital forensic stakeholder roles.
2. Address domain complexity by providing a top-down view and to find commonality patterns across implementations.
3. The models are extensible to address the introduction of different implementations and implementation types.
4. Construct a digital forensic domain specific UML Profile for digital evidence creation which can be used for future modeling efforts.

The animated visualization application utilizing the behavioral script of the executable model addressed the first objective. The application showed information at varying levels of detail for various scenarios. DFCS staff reviewed and commented on the visualization application. The outcome of the review was that the model-based application could be useful for understanding to the stakeholders in an educational environment. The application was able to show various levels of detail, however, the
expert review brought up concerns with how the application would support the analyst and other stakeholder roles.

Identifying common patterns across implementations addressed the second objective. These patterns suggested that there were commonalities across the actual implementations of components and devices which are of interest to digital forensic stakeholders. These patterns were used in the development of the top-level models for the three use-case scenarios. The abstraction level of the top-level model was quantified in metrics. The top-level models provided an abstract description of the different implementations. The pattern commonality within a use-case scenario and across use-case scenarios was quantitatively captured in the metrics.

A repeatable, documented, procedure to extend the top-level models and digital forensic profiles addressed the third objective. Extending the top-level model was verified by adding implementations to the use-cases. Extending the top-level profile was verified by combining the profiles of each use-case.

The digital forensic profile addresses the forth objective. This UML profile, if utilized, would support future digital forensic modeling efforts.

5.3 Limitations
This work only demonstrated utility for one stakeholder group, educators. The application developed for this work contained features to provide additional details and some additional features targeted to the analyst role. However, feedback from expert reviewers in the analyst community indicated that it was not clear how this particular type of application would be useful for the analyst or other stakeholder roles.
This work utilized open source tools. During the timeframe of this research was being conducted, xtUML had a significant pedigree and support for the development of executable models. However, xtUML is not strictly UML compliant and did not support the requisite UML features required for this work. As an example, it does not support OCL and profiles. To address the need to develop profiles, a second environment, Papyrus, which is UML 2.5 compliant was used for the profiling aspects of this work, meaning that two modeling environments were used in this work. However, the two Open Source projects have started collaborating to develop a Papyrus-xtUML environment.

The utility of the top-level models or the utility of profiles was not evaluated with digital forensic stakeholders. This work focused on identifying why and how software engineering modeling concepts could be employed to digital forensics. Assessing the potential utility and impact of applying modeling in the digital forensic domain was beyond the scope of this work.

There are a combinatorically explosive number of digital forensic scenarios along with a practically uncountable number of configurations. As such, this number of scenarios addressed by this work was infinitesimally small. This work focused on the computational mechanisms within one device in one forensic area at a time. It did not investigate more complex scenarios utilizing multiple devices. In addition, addressing virtual environments would be of interest. Additional work will need to be completed to determine whether the approach in this work can be scaled up to address additional implementations, use-case scenarios, and forensic areas.

The implemented application only showed one aspect of how modeling can be
used from an application perspective, a simple animation. Implementing a more sophisticated animation on additional model scenarios would provide additional insight into the utility of this approach.

The focus of this work was to show how evidence was created. There could be other aspects of digital forensics in which modeling could be used. For example, the information in the model could be used to reason about evidence validity.

Patterns were only touched on in this work and only simple patterns were identified. To gain a better understanding of commonality, all significant data structures used in implementations would need to be represented in a model.

The metric results are suggestive that commonality and complexity are inversely proportional. This result was not rigorously proven.

5.4 Next Steps

This work provided a starting framework for applying software engineering modeling techniques to digital forensics. The next steps would include:

- Assess whether the constructive method scales up.
- Evaluate the utility of top-level models and profiles with digital forensic stakeholders.
- Add enhanced visualization techniques to target specific stakeholders.
- Determine if modeling can support other types of applications.
- Validate that complexity and commonality are inversely proportional.
- Address additional stakeholder roles.
A variety of additional use-cases would need to be modeled to assess whether the approach scales-up. The constructive method could be assessed for extensibility as follows:

- Add additional forensic areas and associated use-cases,
- Add implementations to an existing use-case scenario, and
- Model a different level of abstraction for an existing use-case.

As an example, a cloud based scenario involving a combination of network analysis may be of interest, since it would involve multiple devices and technology areas (e.g., network, wireless, cloud based services, virtualization). Adding additional implementations to existing use-case scenarios would provide additional insights on impacts to the use-cases top-level model. For example, adding additional file system implementations (e.g., HFS+, HPFS, UFS, VMFS, ZFS) to the file allocation/delete scenario could further evolve the top-level view. Adding a different level of abstraction to an existing use-case would provide insights into how additional stakeholders could utilizing modeling.

Given the top-level models and resultant profiles generated from this work or follow-on top-level models and profiles developed by assessing the constructive method, the utility of the top-level models and profiles will need to be addressed. The utility could be addressed in an educational scenario or in using the profiles to generate digital forensic models.

Further investigation of how applications could facilitate understanding for digital forensic stakeholders is warranted. As an example, more advanced
visualization techniques may have utility. In addition, there may be other applications of modeling, other than understanding the who, what, when, where, and how of evidence creation may be of interest. For example, the model provided relationships between elements and from that perspective the model is equivalent to a database. Applications can be developed to provide a front-end query to generate reports of information of interest which may be beneficial in an investigation. Another use is to develop applications utilizing artificial intelligence techniques to reason about the evidence.

The relationship between the TLAM and ICM metrics should be further investigated to determine if the claim that commonality and complexity are indeed inversely proportional. A proof should be provided to substantiate the claim.

All future investigations should reach out to stakeholders to incorporate their perspectives. For example:

- Legal stakeholders, to identify the potential benefit models could provide in presenting a case. In addition, insight could be gained into the ways in which visualization techniques may or may not help in the courtroom.

- Law Enforcement stakeholders (e.g., State Police, detectives), to understand the aspects of the evidence that are important and to understand the types of applications that would be of use for training purposes.

- Educational stakeholders, to gain an understanding of whether models could be tools for educational purposes. Moreover, these individuals could provide evidence to help develop an understanding of whether formalism allows for an exchange of ideas between researchers.
• System Analyst stakeholders, to gain an understanding of whether model formalism could help address their comprehension of the underlying systems or as a means to be able to exchange information. Additionally, these stakeholders could provide valuable insight into the applications that would be of use. Finally, these stakeholders would provide information to help develop an understanding of the authoritative sources that could be utilized.

• Tool Vendors stakeholders, to gain an understanding of whether models could be used as a method to standardize descriptions of digital forensic devices and components. Additionally, these stakeholders could provide information to help to develop an understanding of the utility in developing digital forensic model standards that would be beneficial for the digital forensic tool community.

5.5 Contributions of this work
The work provided insights on how modeling would benefit digital forensic in addressing some of the complexities of the field. The unique contributions of this work consisted of the following:

1. Modeled the computational mechanisms involved in the creation of digital forensic evidence,

2. Generated computable model artifacts to facilitate the execution of a digital forensic relevant animation application,

3. Identified a process to construct top-level implementation views of computational mechanisms,
4. Identified modeling patterns to catalog commonalities in computational mechanisms,

5. Identified metrics to support commonality and abstraction analysis, and

6. Introduced a digital forensic profile along with a process to extend the profile.

The models developed represented the computational mechanisms for the creation of digital forensic evidence. Since the models were executable, the model’s behavior could be captured and utilized by an application to animate the model’s behavior.

A repeatable constructive methodology was identified to develop and test abstract top-level models from implementation specific models. The constructive method was also utilized to construct profiles. In developing these models, implementation commonality patterns were identified for the modeled computational mechanisms. The ICM and TLAM metrics were introduced to assess commonality and abstract between top-level and implementation specific models.
**Appendix A: Digital Forensic Profile stereotype descriptions**

Table A-1 provides descriptions of the stereotypes utilized in the constructed digital forensic profile.

Table A-1. Digital Forensic profile stereotype description

<table>
<thead>
<tr>
<th>Profile Package</th>
<th>Profile Element</th>
<th>Profile Meta Class/Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>OS Class</td>
<td>Class</td>
<td>OS with no initial connections.</td>
</tr>
<tr>
<td>OS</td>
<td>FileSystemManagerDetail Class</td>
<td>OS to support detail file system models.</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>FileSystemManager Class</td>
<td>OS with directory structure</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>ProcessManager Class</td>
<td>OS with support of processes and process lists</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>IOManager Class</td>
<td>OS with support of network connections.</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>OsUserCall Association</td>
<td>User space call to the kernel.</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>OsKernalCall Association</td>
<td>Kernel calls.</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>ApiCall Association</td>
<td>API call between components, for example a socket call.</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Content Class</td>
<td>Content of file.</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Browser Class</td>
<td>Web browser application.</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Database Class</td>
<td>Database. Contains Tables.</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Table</td>
<td>Class</td>
<td>Database table contains Row(s).</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Application</td>
<td>Row</td>
<td>Class</td>
<td>Contain meta data on row.</td>
</tr>
<tr>
<td>Application</td>
<td>CommandLine</td>
<td>Class</td>
<td>Command line tool to either get process list or open connections</td>
</tr>
<tr>
<td>Application</td>
<td>Document</td>
<td>Class</td>
<td>Application whose content is stored in a file.</td>
</tr>
<tr>
<td>Application</td>
<td>General Application</td>
<td>Class</td>
<td>Application whose process and network connections are of interest.</td>
</tr>
<tr>
<td>Application</td>
<td>Application</td>
<td>Class</td>
<td>General Application.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>FSMetaStore</td>
<td>Class</td>
<td>Contains the file system meta data.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>DirectoryRoot</td>
<td>Class</td>
<td>Contains other directories or directory entries. Top level directory is the “root&quot;.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>DirectoryEntry</td>
<td>Class</td>
<td>Contains the meta information for the files or subdirectories.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>FileDataUnitRef</td>
<td>Class</td>
<td>A set of references to data unit(s) where the file content is written.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>DUAllocationMechanism</td>
<td>Class</td>
<td>Data structure which keeps track of available data units.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>AllocationCells</td>
<td>Class</td>
<td>An individual cell/element in the DUAllocationMechanism.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>DataUnitManagement</td>
<td>Class</td>
<td>The mechanism which manages the access to the file system data units.</td>
</tr>
<tr>
<td>FileSystem</td>
<td>DataUnit</td>
<td>Class</td>
<td>An individual data unit which can be written to. Typically a cluster or block.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>RAM</td>
<td>Socket</td>
<td>Class</td>
<td>Provides API for application to receive and send information.</td>
</tr>
<tr>
<td>RAM</td>
<td>CommsSocketObject</td>
<td>Class</td>
<td>Represents the network stack between the application sockets to the network “wire”.</td>
</tr>
<tr>
<td>RAM</td>
<td>ConnectionManagement</td>
<td>Class</td>
<td>Maintains a list of network connections.</td>
</tr>
<tr>
<td>RAM</td>
<td>Process</td>
<td>Class</td>
<td>An individual processes. Contains the forensic information of interest.</td>
</tr>
<tr>
<td>RAM</td>
<td>FileDescriptorTable</td>
<td>Class</td>
<td>Contains the network handles for the process of interest.</td>
</tr>
<tr>
<td>RAM</td>
<td>FileDescriptor</td>
<td>Class</td>
<td>Individual file handle for a network object.</td>
</tr>
<tr>
<td>DFEvidenceTypes::RAM</td>
<td>GenericProcess</td>
<td>Tag</td>
<td>Evidence properties associated with a generic process.</td>
</tr>
<tr>
<td>DFEvidenceTypes::RAM</td>
<td>GenericNetwork</td>
<td>Tag</td>
<td>Evidence properties associated with a generic network aware process.</td>
</tr>
<tr>
<td>DFEvidenceTypes::FileSystem</td>
<td>DirectoryEntry</td>
<td>Tag</td>
<td>Evidence properties associated with directory entries.</td>
</tr>
<tr>
<td>DFEvidenceTypes::FileSystem</td>
<td>Directory</td>
<td>Tag</td>
<td>Evidence properties associated with directory.</td>
</tr>
<tr>
<td>DFEvidenceTypes::FileSystem</td>
<td>FSMetaData</td>
<td>Tag</td>
<td>Evidence properties associated with file system meta data.</td>
</tr>
<tr>
<td>DFEvidenceTypes::Browser</td>
<td>CompositeBrowser</td>
<td>Tag</td>
<td>Composite of browser evidence properties.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>-----</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>DFEvidenceTypes::Browser</td>
<td>Downloads</td>
<td>Tag</td>
<td>Evidence properties associated with browser downloads.</td>
</tr>
<tr>
<td>DFEvidenceTypes::Browser</td>
<td>History</td>
<td>Tag</td>
<td>Evidence properties associated with browser visits (history).</td>
</tr>
<tr>
<td>DFEvidenceTypes::Browser</td>
<td>Cookies</td>
<td>Tag</td>
<td>Evidence properties associated with browser cookies.</td>
</tr>
<tr>
<td>DFTypes::FileSystem</td>
<td>OSData</td>
<td>Tag</td>
<td>Supports File System stereotypes.</td>
</tr>
<tr>
<td>DFTypes::FileSystem</td>
<td>FSStructure</td>
<td>Tag</td>
<td>Supports File System stereotypes.</td>
</tr>
<tr>
<td>DFTypes::FileSystem</td>
<td>Header</td>
<td>Tag</td>
<td>Supports File System stereotypes.</td>
</tr>
<tr>
<td>DFTypes::OS</td>
<td>OSSstatus</td>
<td>Tag</td>
<td>Supports OS stereotypes.</td>
</tr>
<tr>
<td>DFTypes</td>
<td>StatusVariables</td>
<td>Tag</td>
<td>Supports OS and application stereotypes.</td>
</tr>
<tr>
<td>DFTypes::Application::Browser</td>
<td>BrowserStatus</td>
<td>Tag</td>
<td>Properties associated with browser status.</td>
</tr>
<tr>
<td>DFTypes::Application::Browser</td>
<td>FileData</td>
<td>Tag</td>
<td>Properties associated with file system support of browser application.</td>
</tr>
<tr>
<td>DFTypes::Application::Browser</td>
<td>RowData</td>
<td>Tag</td>
<td>Properties associated with db table support of browser application.</td>
</tr>
<tr>
<td>DFTypes::Application::FileSystem</td>
<td>AppData</td>
<td>Tag</td>
<td>Properties associated with generic file system for browser application.</td>
</tr>
<tr>
<td>DFTypes::Application::RAM</td>
<td>AppData</td>
<td>Tag</td>
<td>Properties for generic applications.</td>
</tr>
<tr>
<td>DFTypes::Application</td>
<td>Contents</td>
<td>Tag</td>
<td>Properties for file contents.</td>
</tr>
<tr>
<td>DFTypes</td>
<td>DFTime</td>
<td>Tag</td>
<td>Properties for time. Supports numerous stereotypes.</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>-------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>DFTypes</td>
<td>DFEnums</td>
<td>Tag</td>
<td>Enumerations supporting use-cases</td>
</tr>
</tbody>
</table>
Appendix B: Formalism and model aware languages

Basic Theory. The theoretical basis for this work is based on sets and the relationship between sets. The theory in this section is taken directly from [71].

Sets can be mathematically defined using set comprehension. Set comprehension takes the form of:

\[ \{ x : S | P(x) \} \]

This notation says a new set is defined by elements \( x \) which are drawn from domain \( S \) where the elements satisfy a predicate, \( P(x) \), such that it is evaluated to be true. A subset consist of elements drawn from another set:

\[ X \subseteq Y \equiv ( \forall e | e \in X \rightarrow e \in Y) \]

A power set (e.g. \( P(S) \)) is the set of all subsets as defined by:

\[ P(S) \equiv ( e | e \subseteq S) \]

A product of two sets, is the set of all possible pairs \( (x,y) \) where \( x \in X \) and \( y \in Y \) and can be described by set comprehension as:

\[ X \times Y \equiv \{(x,y) | x \in X \land y \in Y\} \]

A binary relation, \( R \), between sets \( X \) and \( Y \) is a subset of the product of two sets:

\[ R \subseteq X \times Y \]

For a given relation, the set of 1st elements is the domain of definition and the set of second elements is the image. Two relations can be mapped to create one relation through relation composition. For example given \( s:AxB \) and \( r:BxC \) then \( s;r : A \times C \). More formally:
\[ s; r \equiv (x, z) | (x, y) \in s \land (y, z) \in r \]

**Model Aware Languages.**

The languages of interest, OAL and OCL are model aware in that they can access the model elements. These languages refer to the elements of a class diagram. The modeling of this work consists of navigating across class or collection class elements to either read or modify attributes. Based on the attribute values the control flow is identified for the next navigation.

OCL is not a programming language but a specification language. OCL can to change the model, it can only return values. OCL is used as either a query language or to specify invariants on classes.

Action languages, such as, OAL are able to instantiate classes, changes class attributes, control state transitions and provide rudimentary constructs to implement functions. The action language can provide the control flow.

**Set Comprehension and Relations.** A UML class is analogous to the formal concept of set comprehension. A UML class provides a template in which objects are instantiated, see Figure B-1. The objects are analogous to the elements being select from a domain. The objects by definition will have all the predefined attributes of a class.
An UML association is analogous to the formal concept of a relation. The association shows some type of relationship between two classes and their associated objects. The association contains the arrow which shows how the association is being navigated. The objects of the class in which is being navigated from contain the objects of the domain of definition. The class which is being navigated to are the image. The multiplicity of the association from the source class will be 1 and on the target class will be either 1 or * (meaning 0 or more). In the example, the association can be represented in the relation notation as:

\[ \{(x_1, y_1), (x_1, y_2), (x_1, y_3)\} \]

Where \(x_1\) is the object being navigated from and \(y_1, y_2\) and \(y_3\) are the objects being navigated to.

**Navigation and selection (OCL and OAL).** Figure B-2 shows and example of navigation across classes. Starting at class \(X\) and navigating to class \(Y\) over relation s
and then navigating from class Y to class Z utilizing relation \( r \), the relation composition would be represented as:

\[
S; r \equiv (x_1, z_1) | (x_1, y_1) \in S \land (y_1, z_1) \in r
\]

Figure B-2. Navigation example.

In navigation it is often needed to select a particular object in a collection. For example, see Figure B-3. In this example, the starting context is class X and the objective is to select the object from class Z where key has a value of two. See Figure B-4 for the OCL representation.
Figure B-3. Navigation and selection example.

\[ R = \{(x_1, y_1), (y_1, z_1), (y_1, z_2), (y_1, z_3)\} \]

Figure B-4. OCL Navigation and Selection Example.

The equivalent OAL representation is shown in Figure B-5 and the OAL source code representation is as follows:

```oal
select one z related by x->Y[R1]->Z[R2] where selected.key == 2;
```

Figure B-5. OAL Navigation and Selection Example
BIGLIOGRAPHY


[14] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.


[43] D. Silingas, R. Vitutinas, A. Armonas and L. Nemuralite, "Domain-Specific Model Environment Based on UML Profiles".


